P. 1847/91

1
1991

informatyka

Komputerowa analiza obrazów w biologii i medycynie

Współczesne metody kompresji danych

WYDAWNICTWO NOT SIGMA

Cena 5800 zł

LET C = NOT C OR C.

Ciągle zdumiewa mnie znaczna popularność języka C. Nie dziwię się tutaj naszym informatykom, którzy dla utrudnienia sobie życia, ale dla jego równoczesnego uprzyjemnienia wchłaniają nowinki zza Atlantyku. Moje zdziwienie budzi miłość do C w Stanach, przy czym biorąc proporcjonalnie do liczby programistów, liczba fanów i bezpośrednich użytkowników C jest tam znacznie mniejsza niż u nas w kraju. Ja uważam (patrz przypis na końcu), że popularność tego języka jest przesadzona w stosunku do jego rzeczywistej wartości.

Aha, znalazł się ten, który przyzwyczaił się do programowania w BASICu, powie ktoś. Zanim więc postaram się uzasadnić moją awersję do C, podam parę faktów z mojego życiorysu programisty. W ciągu ponad 15 lat poznałem, programowałem i nauczałem sześć języków programowania wysokiego poziomu (Algol-60, BASIC, Fortran, Pascal, Modula-2, Ada) oraz dwa makroasemblery (MACRO-11 i MASM). W zeszłym roku przez osiem miesięcy bardzo intensywnie programowałem w języku C (Microsoft, wersja 5.0). Mam też trochę wiedzy na temat teorii struktur danych oraz konstrukcji oprogramowania systemowego.

Język C jest językiem niskiego poziomu przeznaczonym do konstruowania programowania systemowego, ale według jego autorów nadaje się również do pisania programów numerycznych, przetwarzania tekstów czy baz danych. W reklamach i opisach tego języka pojawiają się też następujące stwierdzenia:

1. *Łatwość poznania i przyswojenia.*
2. *Nowoczesne struktury danych.*
3. *Nowoczesne instrukcje strukturalne.*
4. *Użyteczne operacje na danych różnych typów.*
5. *Duża liczba operatorów.*
6. *Możliwość prostej realizacji operacji maszynowych.*
7. *Zwartość zapisu tekstu programu.*
8. *Efektywność i mały rozmiar programów.*
9. *Bogata biblioteka przenośnych podprogramów standardowych.*
10. *Łatwa przenośność programów.*

Wszystkie te hasła są w zasadzie prawdziwe, ale czy aby na pewno zawsze trzeba się z nimi zgodzić?

Podstawowy opis języka jest faktycznie bardzo prosty. Już po pobieżnym nawet przejrzeniu dowolnego opisu możemy przystąpić do napisania prostych programów. Problemy zaczynają się w przypadku chęci zapisania nieco bardziej złożonych algorytmów, szczególnie typu kombinatorycznego. W tym momencie musimy bowiem biegle opanować zasady wskazywanego odwoływania się do zmiennych prostych czy struktur deklarowanych statycznie lub dynamicznie. Nawet tak sympatyczny typ, jak tablica, staje się niekiedy mało zrozumiały.

Zacznijmy zatem od deklaracji tablicy dwuwymiarowej:

```
int day_tab[2] [13];        (* a *)        [KR, 123]
int day_tab[13]             (* b *)
int (* day_tab[13];         (* c *)
int * day_tab[13];          (* d *)
```

Deklaracja 'a' jest typowa. W deklaracji 'b' okazuje się, że liczba wierszy jest nieistotna, a więc można ją pominąć. Deklaracja 'c' określa wskazanie na tablicę jednowymiarową z 13 liczbami całkowitymi, a deklaracja 'd' tablicę z 13 wskazaniami na liczby całkowite. Jednym tchem nie można tego przeczytać. Patrząc na ten przykład postanowiłem określić możliwe nazwy elementu tablicy dwuwymiarowej:

```
int table [2][13];
tab[1][2]          tab[0][15]          (*(tab+1))[2]
*(*(tab+1)+2)      (*(&tab[0]+1))[2]   (*(&tab[1]))[2]
(*&tab) [1][2]     1[tab][2]           0[tab][15]
```

Wszystkie nazwy dotyczą tego samego elementu. Muszę się tutaj przyznać, że poza prostymi przypadkami, pozostałe tworzyłem z pomocą kompilatora Turbo C. Stawiam teraz pytanie, czy jest to proste nawet dla doświadczonego programisty?

Trzeba jednak dodać, że przy niezłej znajomości teorii struktur danych, zrozumienie tych konstrukcji jest znacznie prostsze. Stwierdzenie to dotyczy również pozostałych struktur danych, które są podobne do struktur z innych języków. Czy są one nowocześniejsze, trudno powiedzieć. W każdym razie stale mam wrażenie, że deklaracja typedef została wprowadzona znacznie później, trochę przypadkowo. Jej zrozumienie w powiązaniu ze składnią struktury struct oraz możliwościami # define nie jest całkiem proste.

Zestaw instrukcji jest w języku C faktycznie bardzo efektywny. Ale i tutaj można mieć trochę zastrzeżeń, na przykład do składni instrukcji

```
for (i=1; i<n; i++);
```

w której trzecie wyrażenie jest wykonywane wewnątrz cyklu. A jeszcze dodatkowa możliwość wpisania kilku instrukcji wewnątrz opisu cyklu może czasem przysporzyć kłopotów. Łatwo też zgubić treść cyklu domykając średnikiem za wcześnie nagłówek instrukcji. Kompilator z reguły nie zasygnalizuje tego przypadku.

dokończenie na s. 23

PRZEDSIĘBIORSTWO ZASTOSOWAŃ INFORMATYKI

meditronik

oferuje:

- systemy mikrokomputerowe
- programy aplikacyjne dla różnych dziedzin gospodarki (na życzenie wysyłamy katalog)
- poszukiwane komponenty elektroniczne
- interfejs do kamery video (opc. CCD) z bogatą biblioteką oprogramowania
- emulator Z80
- tester układów scalonych i pamięci
- programator BPROM
- asynchroniczny procesor komunikacyjny
- konwerter RS-232 – Centronics

instaluje:

- połączenia międzykomputerowe (XT/AT – ODRA/RIAD/IBM)
- systemy sieciowe (NOVELL)
- systemy wielodostępne (SCO Xenix 286, 386, Unix System V)

Jeżeli jesteś autorem oryginalnego programu aplikacyjnego

- skontaktuj się z nami, będziemy pośredniczyć w sprzedaży Twojego programu, dbając o ochronę Twoich praw autorskich!

Nasz adres:

00–194 Warszawa, ul. Dzika 4
tel. (02) 635–22–63, 635–22–64,
fax (02) 635–21–95
tlx 816075 medi pl

EO/605/89

informatyka

nr 1 1991
styczeń
Rok wydania XXVI

Miesięcznik

ISSN 0542-9951. INDEKS 36124

KOLEGIUM REDAKCYJNE:

mgr Jarosław DEMINET
dr inż. Wacław ISZKOWSKI
mgr Teresa JABŁOŃSKA
(sekretarz redakcji)
Władysław KLEPACZ
(redaktor naczelny)
dr inż. Wojciech MOKRZYCKI
mgr inż. Jan RYŻKO
mgr Hanna WŁODARSKA

PRZEWODNICZĄCY
RADY PROGRAMOWEJ:

Prof. dr hab.
Juliusz Lech KULIKOWSKI

WYDAWCA: Wydawnictwo Czasopism i Książek Technicznych SIGMA NOT
Spółka z o.o.
ul. Biała 4,
00-950 WARSZAWA
skrytka pocztowa 1004

Redakcja: 01-552 Warszawa,
Pl. Inwalidów 10 p. 104, 105
tel. 39-14-34

Materiałów nie zamówionych redakcja nie zwraca

DRUK:
PRASOWE ZAKŁADY GRAFICZNE
RSW „PRASA-KSIĄŻKA-RUCH"
ul. Dworcowa 13,
85-950 BYDGOSZCZ
zam. 3293/90
Obj. 3,0 ark. druk. Nakład 3650 egz.

Cena egzemplarza 5800 zł

W sprawach ogłoszeń prosimy zwracać się bezpośrednio do Redakcji

W numerze:

	Strona
Komputerowa analiza obrazów w biologii i medycynie – *Juliusz Lech Kulikowski*	1
Współczesne metody kompresji danych: teoria, algorytmy i zastosowania – *Wojciech Skaba*	6
System CASE – przykłady narzędzi (2) – *Piotr Fuglewicz*	14
/unix	
O standardzie ANSI języka C (1) – *Władysław Majerski*	18
Plan 9, czyli ucieczka do przodu – *Jacek Surma*	20
Ze świata	
CeBIT '90 – *Władysław Klepacz*	21
zBITki	
LET C = NOT C OR C. – *Wacław Iszkowski*	III s.okł.

W najbliższych numerach:

- **Henryk Budzisz charakteryzuje systemy ekspertowe używane do projektowania układów cyfrowych.**
- **Jarosław Deminet dzieli się wrażeniami z wizyty w Kongresie USA, której celem było poznanie organizacji pracy biur Kongresu, ze szczególnym uwzględnieniem wykorzystania technik informacyjnych i komputerowych.**
- **Jan Ryżko opisuje skanery optyczne wykorzystywane do odczytu rysunków architektonicznych i technicznych.**
- **Wacław Iszkowski zajmuje się problemem reprezentacji w komputerowym przetwarzaniu informacji krajowego zestawu znaków liter alfabetu polskiego.**

Warunki prenumeraty

Przyjęcie prenumeraty – wyłącznie na podstawie dokonanej wpłaty na drukach dostarczanych dotychczasowym prenumeratorom przez Wydawnictwo, lub nowym – po uprzednim zgłoszeniu zapotrzebowania (pisemnie lub telefonicznie) w Zakładzie Kolportażu Wydawnictwa.

Blankiet wpłaty – powinien zawierać następujące informacje: dokładna nazwa i adres (z kodem pocztowym) zamawiającego, tytuły zamawianych czasopism, ich liczbę i okres prenumeraty.

Wpłata – zgodnie z podanymi cenami należy dokonać w banku lub w UPT na konto podane na naszym blankiecie, tj.:

Państwowy Bank Kredytowy III O/Warszawa nr: 370015-1573-139-11

Prenumeratorzy zbiorowi – osoby prawne obowiązują blankiety „Wpłata-Zamówienie". Cena normalna.

Prenumeratorzy indywidualni – osoby fizyczne obowiązują blankiety typu przekazy dla wpłat na rachunki bankowe. Cena normalna.

Prenumerata ulgowa – zgodnie z podaną ceną ulgową przysługuje wyłącznie osobom fizycznym, będącym członkami SNT, studentom i uczniom szkół zawodowych. Uczniowie szkół ogólnokształcących mogą zamówić w prenumeracie ulgowej tylko miesięcznik „Aura". Uwaga: w podanym okresie prenumeraty można zamówić tylko po jednym egzemplarzu z każdego tytułu.

Prenumerata ze zleceniem wysyłki za granicę – cena prenumeraty jest dwukrotnie wyższa od ceny normalnej. Należy podać dokładny adres odbiorcy za granicą.

Terminy przyjmowania prenumeraty:
do 10 listopada na I, II, III, IV kwartał następnego roku
do 28 lutego na II, III, IV kwartał br.
do 31 maja na III, IV kwartał br.
do 31 sierpnia na IV kwartał br.
Zmiany w prenumeracie, np. zmiana liczby tytułów, liczby egzemplarzy, rezygnacja z prenumeraty, można zgłaszać tylko w podanych terminach z mocą obowiązującą od następnego kwartału.

Egzemplarze archiwalne (z lat ubiegłych)
Można nabyć za gotówkę w Klubie Prasy Technicznej w Warszawie, ul. Mazowiecka 12 (tel. 26-80-16) lub zamówić pisemnie w Zakładzie Kolportażu, Dział Handlowy, 00-950 Warszawa skr. poczt. 1004 (tel. 40-37-31) na rachunek lub za zaliczeniem pocztowym.

Informacji o prenumeracie udziela: Zakład Kolportażu Wydawnictwa SIGMA-NOT Spółka z o.o., 00-716 Warszawa, ul. Bartycka 20, skr. 1004. Telefony: 40-00-21 wewn. 293, 295, 299 lub 40-30-86, 40-35-89.

Wstępna cena jednego egzemplarza na 1990 rok: normalna – 5800 zł
ulgowa – 2900 zł

Wartość prenumeraty (w zł):
Normalna: kwartalna – 17 400, półroczna 34 800, roczna 69 600
Ulgowa: kwartalna – 8700, półroczna 17 400, roczna 34 800

Uwaga: W przypadku zmiany cen w okresie objętym prenumeratą, prenumeratorzy zobowiązani są do dopłaty różnicy cen.

P.1847/91

JULIUSZ LECH KULIKOWSKI
Instytut Biocybernetyki
i Inżynierii Biomedycznej PAN
Warszawa

Komputerowa analiza obrazów w biologii i medycynie

Zainteresowanie komputerem jako narzędziem umożliwiającym automatyczną analizę obrazów sięga końca lat pięćdziesiątych. Początkowo ograniczało się ono do jednego tylko aspektu takiej analizy: rozpoznawania prostych znaków graficznych. Wynikało to z próby znalezienia alternatywnej drogi do rozwiązania problemu rozpoznawania względem tej, którą wytyczyły prace F. Rosenblatta związane z budową perceptronu [4, 7]. Rosenblatt starał się stworzyć model fizyczny, który wyjaśniałby zasady działania kory mózgowej człowieka, a zwłaszcza jej fragmentu odpowiedzialnego za przetwarzanie obrazów wzrokowych. Komputerowa analiza obrazów odeszła jednak od tych założeń i przyczyniła się do rozwoju rozpoznawania obrazów jako odrębnej dziedziny badań, bardziej związanych z cybernetyką niż z neurofizjologią. Wkrótce w rozpoznawaniu obrazów zaczęto powszechnie stosować z jednej strony doświadczenia i środki techniczne zaczerpnięte z techniki telewizyjnej i komputerowej, z drugiej zaś – modele teoretyczne i metody zapożyczone ze statystyki matematycznej, analizy funkcjonalnej, geometrii wielowymiarowej, algebry itp.

W latach sześćdziesiątych badania nad rozpoznawaniem obrazów podjęło też kilka ośrodków w Polsce. Jednym z bardziej znanych stał się np. system CPO-1/ODRA 1204 zbudowany w ówczesnym Instytucie Automatyki PAN w Warszawie [5]. System ten stał się zaczątkiem prac konstrukcyjnych nad następnymi systemami oznaczonymi podobnym skrótem: CPO – Cyfrowy Przetwornik Obrazów. Wykorzystano w nich bardziej nowoczesne jednostki przetwarzające. Tak powstały: CPO-2/K-202 i jego mutacja CPO-2/MERA 400, oparte na minikomputerach produkowanych w Polsce w latach siedemdziesiątych. Systemy te posłużyły do pierwszych badań nad komputerowym przetwarzaniem obrazów biologicznych. Szybko przekonano się, że jest to problem nieporównanie trudniejszy niż rozpoznawanie znaków graficznych we wcześniej prowadzonych pracach. Warto dodać, że jako pierwszy ideę badań nad rozpoznawaniem obrazów w IA PAN podjął mgr inż. Jacek Karpiński, a w późniejszej fazie prac duży wkład do badań wnieśli m.in. inż. W. Jarosiński, mgr inż. J. Dernałowicz, dr inż. Z. Kulpa, mgr Maria Piotrowicz i mgr inż. M. Chmielewski. Mimo licznych reorganizacji zespół ten i jego podstawowy kierunek badawczy udało się utrzymać i dziś; w znacznie rozszerzonym składzie działa on w Instytucie Biocybernetyki i Inżynierii Biomedycznej PAN.

Postęp w dziedzinie komputerowego rozpoznawania (lub szerzej – komputerowej analizy obrazów) od lat napotykał barierę, którą wyznaczał stan rozwoju techniki komputerowej, w miarę zaś jak barierę tę przesuwano, występowała (i daje się odczuć obecnie) nowa bariera – techniki szybkiej rejestracji w postaci cyfrowej obrazów o wysokim standardzie, przekraczającym standard telewizji abonenckiej. Kiedy i ta bariera zostanie przezwyciężona, co jest w pełni realne, pojawi się następna – kto wie, czy nie najtrudniejsza do pokonania – bariera umiejętności konstruowania algorytmów efektywnej analizy obrazów, uwzględniającej ich zawartość treściową.

Obrazy biologiczne

W rozumieniu fizycznym obraz jest sygnałem optycznym mieszczącym się w określonym pasmie częstotliwości (nie koniecznie widzialnym), zależnym od argumentów przestrzennych, którego organizacja odwzorowuje rozkład przestrzenny określonego czynnika fizycznego, ujawniającego się przez samorzutne promieniowanie, albo też pochłanianie, odbijanie lub rozpraszanie promieniowania pochodzącego ze źródeł zewnętrznych. W formowaniu obrazu biorą zatem udział co najmniej dwa czynniki: źródło lub źródła promieniowania oraz ów czynnik przewodzący (pochłaniający) lub rozpraszający promieniowanie. Tylko jeden z tych czynników stanowi przedmiot zainteresowania obserwatora, tj. przenosi informację użyteczną, drugi natomiast jest czynnikiem informacyjnie biernym. W praktyce zwykle towarzyszą im jeszcze czynniki trzeciego rodzaju: zniekształcające lub zakłócające obraz na skutek dodatkowych odbić, załamań, pochłaniań lub rozszczepień promieni. Wyodrębnienie w sygnale czynników czynnych (przenoszących właściwą informację), biernych (nie przenoszących informacji właściwej, ale mogących ułatwić odbiór informacji, jeśli czynniki te są znane), a wreszcie czynników zakłócających – nie różni analizy obrazów od innych dziedzin przesyłania lub przetwarzania informacji, choć natura fizyczna sygnałów może być różna. Przykładowo, w analizie obrazu mikroskopowego czynną rolę odgrywa przestrzenne rozmieszczenie czynnika pochłaniającego promienie świetlne, utożsamiane z kształtem badanego obiektu (na przykład, komórki biologicznej), a czynnikiem biernym jest skala powiększenia optycznego, widmo promieniowania oświetlającego itp. Czynnikiem zakłócającym może być każdy inny obiekt utrudniający obserwację – zawiesina w płynie zawierającym preparat, refleks świetlny itp. Podobnego rozróżnienia czynników można dokonać w odniesieniu do analizy obrazów rentgenologicznych, ultrasonograficznych, scyntygraficznych, NMR-owskich (opartych na jądrowym rezonansie magnetycznym) itp. W komputerowej analizie obrazów, podobnie jak w innych dziedzinach przetwarzania informacji, zmierza się do tego, by w pierwszej fazie przetwarzania wyeliminować czynniki zakłócające i wydobyć z obrazu składniki zawierające informację użyteczną, wykorzystując do tego, tak dalece jak to jest możliwe, informację o czynnikach biernych.

Obrazy, pojawiające się przy obserwacji obiektów biologicznych: organelli wewnątrzkomórkowych, komórek biologicznych, tkanek, narządów, tworów powstałych na skutek zmian patologicznych w narządach lub tkankach, a na wyższych piętrach obserwacji – obrazy przedstawicieli określonego gatunku biologicznego, ich zachowania się w populacji albo obrazy osobniczego rozwoju itp., mają jednak cechy specyficzne, które w istotny sposób wpływają na możliwości ich komputerowej analizy. I tak:

- obiekty biologiczne są często amorficzne, trudno opisywalne w języku geometrii;
- w wielu przypadkach obiekty biologiczne są obserwowane w ruchu;
- wpływ ubocznych czynników obserwacji (czynników biernych) na obserwowaną postać obrazu jest bardzo silny;
- niektóre cechy obrazów, istotne dla analizy, leżą na granicy dostrzegalności, określonej przez parametry techniczne aparatury.

Język biologii lub anatomii zawiera wiele takich terminów, jak *ciałko, wypustka, osłonka, pęczek*, których geometryczne znaczenie jest bardzo trudno sprecyzować. Co więcej, owe *ciałka* i *wypustki* z każdą chwilą zmieniają swój kształt i położenie. Powstaje zatem problem: jak komputer może je na obrazie rozpoznać lub na serii obrazów tego samego obiektu – identyfikować? Dochodzą do tego następne trudności: badany obiekt staje się w pełni lub częściowo widoczny i ma określony kształt w pewnych warunkach obserwacji, jednak przy zmianie tych warunków jego kształt ulega zmianie albo obraz częściowo lub całkowicie znika. Jedynie duże doświadczenie i wiedza medyczna pozwala lekarzowi prawidłowo zinterpretować obraz rentgenograficzny lub ultrasonograficzny, ale kryteria, jakimi się przy tym posługuje, nie poddają się łatwej formalizacji.

Problem znalezienia wspólnego języka i wspólnych kryteriów ocen przez informatyka i lekarza lub biologa jest jednym z najtrudniejszych.

cz. 19.858

Informatyk może bez trudu zaproponować algorytm, który będzie – na przykład – owalne kształty okrągłe odróżniał od podłużnych, bez względu na ich położenie i wielkość – jest to jeden z typowych problemów rozpoznawania obrazów. Taki algorytm jednak lekarza najczęściej nie interesuje. Interesuje go natomiast, czy widoczny na obrazie obiekt jest zaczątkiem guza nowotworowego, czy mniej groźnym ogniskiem zapalnym. Na to pytanie informatyk nie ma, oczywiście, gotowej odpowiedzi, wobec czego proponuje uściślenie kryteriów i wyodrębnienie cech, które mogą stać się podstawą klasyfikacji. To żądanie zaskakuje z kolei lekarza, ponieważ umie on rozpoznawać i oceniać obiekty całościowo, bez rozdzielania cech itd. Przy dobrej woli obu stron może dojść do kompromisu: wskazania przez lekarza mierzalnych cech obiektu, które mogą być podstawą do diagnozy oraz zaproponowania przez informatyka:

1) algorytmów oceny (ilościowej lub jakościowej) wskazanych cech oraz
2) metody i algorytmu selekcji spośród wskazanych cech tych, których siła dyskryminacyjna okaże się największa i wystarczająca do diagnozy.

Jeśli uda się to osiągnąć, można to uznać za powód do satysfakcji dla obu stron, lecz nie przesądza to jeszcze o sukcesie nowej metody diagnostycznej.

Przytoczony tu przykład pokazuje, dlaczego w kontekście zastosowań biologicznych lub medycznych mówi się raczej o analizie obrazów niż o ich rozpoznawaniu. Analiza obrazów jest bowiem pojęciem szerszym: zgodnie z klasyfikacją wprowadzoną przez W. K. Giloia [3] oznacza ona taki sposób przetwarzania, w którym obrazowi zostaje przyporządkowany pewien jego opis symboliczny. Opis ten może zawierać po prostu nazwę lub identyfikator klasy obiektów, do której załącza się obiekt widoczny na obrazie – jest to wówczas rozpoznawanie (obrazów, postaci itp.). Może on jednak także zawierać: wartości parametrów kształtu (analiza morfometryczna), wartości współczynników rozkładu w szeregi funkcji ortogonalnych (analiza widmowa), nazwy zaobserwowanych cech jakościowych (analiza jakościowa), wartości parametrów statystycznych (analiza statystyczna), opis cech strukturalnych (analiza strukturalna), wreszcie – opis zawartości z punktu widzenia właściwości obiektu widocznego na obrazie (analiza semantyczna).

Analiza obrazów, zwłaszcza morfometryczna, widmowa lub statystyczna, może stanowić etap przygotowawczy do diagnozy, czyli do właściwego rozpoznania. Takie rozdzielenie diagnozy na etapy jest uzasadnione zwłaszcza wtedy, kiedy w końcowym rozpoznaniu trzeba brać pod uwagę wiele dodatkowych informacji, jak to najczęściej dzieje się w diagnostyce lekarskiej. Diagnozę podejmuje zatem lekarz, korzystając jednak z wyników analizy dostarczonych przez komputer. Takie podejście do komputerowego wspomagania (a nie – automatyzacji) diagnozowania lekarskiego ma następujące zalety:

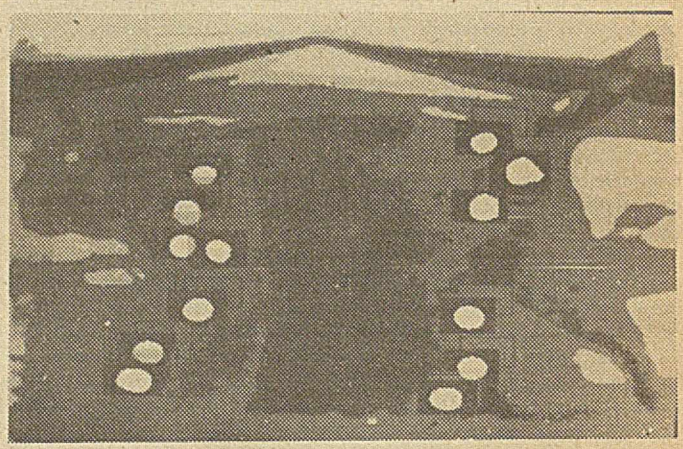
- zapewnia dużą jednoznaczność oceny parametrów będących podstawą diagnozy;
- zmniejsza pracochłonność niektórych, wyjątkowo żmudnych czynności poprzedzających właściwą diagnozę, wykonywanych dotychczas sposobem tradycyjnym;
- zapewnia lekarzowi możliwość indywidualnego dobierania rodzaju badań szczegółowych, którym poddaje się pacjenta, z uwzględnieniem stopnia ich rzeczywistej potrzeby w danym przypadku, ryzyka dla pacjenta, kosztów itp.;
- ułatwia bezpośrednie powiązanie diagnozy z nadzorowaniem procesów terapeutycznych;
- zwiększa odpowiedzialność, ale też – podnosi stopień zaufania pacjenta do lekarza.

Wymagania systemowe

Dopóki komputerowe systemy przetwarzania obrazów były oparte na komputerach drugiej i trzeciej generacji, ich zastosowanie nie wykraczało poza skalę eksperymentalną. Rozwój techniki mikrokomputerowej otworzył przed tą dziedziną nowe perspektywy. Jak już wspomniano, o możliwości praktycznego zastosowania takiego systemu w diagnostyce lekarskiej decyduje nie tylko właściwie dobrany algorytm analizy obrazów, ale także:

- możliwości łatwego połączenia systemu z istniejącą aparaturą służącą do pobierania obrazów (np. z aparaturą rentgenowską),
- łatwość obsługi systemu przez personel medyczny,
- duża niezawodność systemu, dostępność serwisu technicznego,
- koszt systemu (wraz z kosztem urządzeń dodatkowych).

Systemy oparte na mikrokomputerach w dużym stopniu spełniają te wymagania. Co więcej, bogate oprogramowanie użytkowe mikrokomputerów umożliwia rozszerzenie systemów komputerowego przetwarzania obrazów o moduły obliczeń statystycznych, bez danych i prezentacji graficznej, co znacznie zwiększa zakres stosowalności takich systemów.



Rys. 2. Komputerowa analiza ruchów kota

Typowym rozwiązaniem oferowanym dziś przez liczne firmy (także krajowe) jest komputer klasy IBM PC/AT wyposażony w kartę obrazową umożliwiającą wprowadzenie do pamięci obrazowej obrazu z monochromatycznej kamery telewizyjnej (widikonowej lub typu ccd) o standardzie 512 × 512 piksli przy 256 poziomach szarości z częstością 50 obrazów/s. Przeciętny koszt takiego systemu, wraz z biblioteką podstawowych procedur umożliwiających operacje wczytywania i elementarnego przetwarzania obrazów lub ich fragmentów (okien) jest w przybliżeniu dwukrotnie większy niż koszt bazowego systemu mikrokomputerowego. Rodzi się pytanie, czy taki system zaspokaja wymagania, jakie wysuwa diagnostyka lekarska lub praktyka badań biologicznych. Odpowiedź na to nie jest jednoznaczna. Systemy o podanych

a b c

Rys. 1. Przetwarzanie rentgenowskich obrazów naczyń wieńcowych serca: a) obraz tła, b) obraz naczyń wypełnionych środkiem kontrastowym, c) efekt komputerowego odjęcia obrazu a) od obrazu b) (przed filtracją)

parametrach mogą w pełni zdać egzamin przydatności w badaniach morfometrycznych lub statystycznych, wykonywanych na obrazach mikroskopowych, a także przy analizie obrazów radiologicznych uzyskiwanych za pomocą tzw. gamma-kamery, obrazów ultrasonograficznych lub im podobnych, gdyż są to obrazy nie wymagające ani wysokiej zdolności rozdzielczej, ani rozległej skali poziomów cieniowania, ani wreszcie przetwarzania w czasie rzeczywistym z dużą częstością powtarzania.

Odmiennie jednak sprawa przedstawia się w przypadku komputerowej analizy obrazów innego typu. Obrazy NMR-owskie (technika w Polsce mało jeszcze znana, ze względu na jej ogromny koszt) odznaczają się wprawdzie niewielką przestrzenną zdolnością rozdzielczą, ale stawiają duże wymagania kontrastowej zdolności rozdzielczej, czyli liczbie rozróżnialnych poziomów szarości. Rodzi to pewną trudność techniczną, gdyż wymaga opracowania przetworników anologowo-cyfrowych o większej liczbie poziomów kwantowania, a także uniemożliwia zapamiętanie wartości pojedynczego piksla w pojedynczym półsłowie lub rejestrze o pojemności jednego bajta.

Analizie obrazów rentgenowskich stawia się natomiast duże wymagania zarówno odnośnie przestrzennej, jak i kontrastowej zdolności rozdzielczej. Dotyczy to zwłaszcza tzw. angiografii różnicowej, czyli techniki obrazowania naczyń krwionośnych, polegającej na odjęciu od obrazu naczyń wypełnionych środkiem kontrastowym obrazu naczyń bez środka kontrastowego, czyli tzw. tła. Technika ta jest stosowana m.in. w kardiologii, w której służy do badania stanu naczyń wieńcowych serca (koronarografia), jak również do oceny stopnia kurczliwości komór (wentrykulografia). Zdolność rozdzielcza obrazu decyduje w tym przypadku zarówno o rozmiarach naczyń, które można poddać obserwacji, jak i o dokładności pomiarów. Jest to jednocześnie technika, która wymaga analizy długich serii obrazów w czasie rzeczywistym: niektóre rodzaje pomiarów wymagają obserwowania procesu rozpływania się kontrastu w naczyniach wieńcowych w ciągu kilku cykli pracy serca. Jeśli przyjąć, że cykl taki trwa ok. 1 s, to na pełną obserwację może składać się ponad 200 klatek obrazowych. Zakładając, że na zapamiętanie jednej klatki obrazowej o standardzie 512 × 512 piksli i 8 bitów/piksel potrzeba 0,25 MB pamięci, zapamiętanie pełnej serii obrazów w celu poddania ich późniejszej szczegółowej analizie wymaga nie mniej niż 50 MB pamięci. Przy obecnym stanie techniki informatycznej nie są to parametry nieosiągalne, ale też obliczono je dla obrazów o niskim standardzie. Na szczęście procesy fizjologiczne, które rejestruję się dla potrzeb diagnostyki lekarskiej, w większości nie wymagają przetwarzania bardzo intensywnych strumieni informacji. Dotyczą one bowiem albo procesów wolnozmiennych (związanych z krążeniem, oddychaniem, ruchami kończyn, ruchami jelit itp.), albo też procesów szybkozmiennych, ale zawierających niewielką liczbę składowych (np. przy rejestracji krzywych EKG, EEG lub EMG).

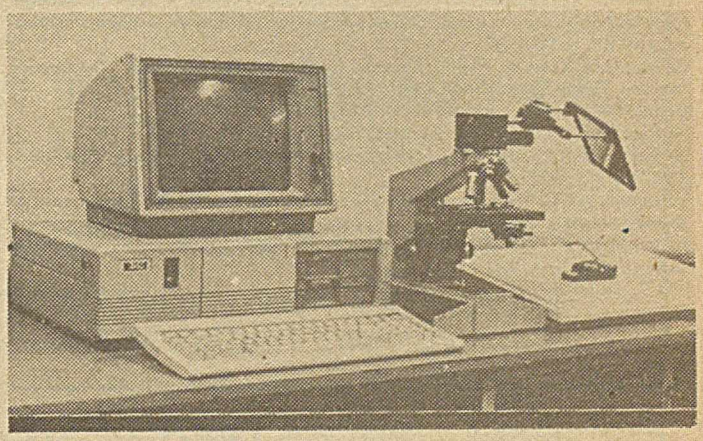
W dotychczasowych rozważaniach nie wspominałem o obrazach barwnych. W większości przypadków obrazy w diagnostyce lekarskiej z natury rzeczy są monochromatyczne, gdyż powstają w wyniku przetworzenia innego sygnału fizycznego – promieniowania rentgenowskiego, ultradźwiękowego itp. W takich przypadkach pewne znaczenie może mieć jedynie sztuczne kolorowanie obrazu monochromatycznego, w celu poprawienia jego czytelności przy obserwacji bezpośredniej. Wbrew oczekiwaniom informatyków takie zabiegi rzadko spotykają się z uznaniem ze strony lekarzy-diagnostyków. Przetwarzanie obrazów barwnych może jednak mieć zastosowanie zarówno w badaniach biologicznych, jak i w medycynie. Połączenie techniki telewizji barwnej z techniką komputerową stwarza bowiem możliwość wykonywania analiz kolorymetrycznych znacznie dokładniej niż nieuzbrojonym okiem. Tak więc barwa może być wykorzystana jako dodatkowa cecha, umożliwiająca opis lub klasyfikację obiektów biologicznych widocznych w ich barwach naturalnych, albo też barwionych sztucznie za pomocą odpowiednich technik laboratoryjnych.

Ciekawą, choć nie opracowaną jeszcze dostatecznie (i nie przez wszystkich lekarzy akceptowaną) techniką diagnostyczną jest irydologia – ocena stanu narządów wewnętrznych pacjenta na podstawie oględzin tęczówki oka. W licznych przypadkach udało się stwierdzić, że zmiany chorobowe zachodzące w narządach wewnętrznych pacjenta znajdują odbicie w rozmieszczeniu pigmentu w tęczówkach oczu. Jeśli obserwacje te znajdą pełniejsze potwierdzenie, będą one mogły być wykorzystane w szybkiej, wstępnej diagnozie lekarskiej, zwłaszcza w tych przypadkach, w których stan pacjenta nie pozwala na nawiązanie kontaktu słownego z lekarzem. Technika komputerowego przetwarzania obrazów barwnych mogłaby w takich przypadkach ułatwić stawianie prawidłowych diagnoz.

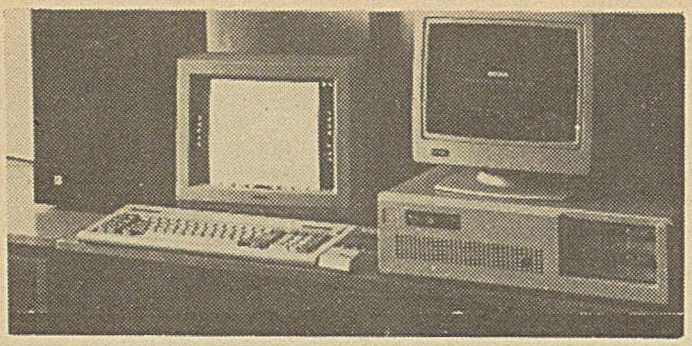
Przetwarzanie obrazów barwnych zwiększy wymagania stawiane systemom komputerowym. Będzie to dotyczyć przede wszystkim pojemności pamięci do przechowywania obrazów. Także zdolność rozdzielcza barwnych kamer tv (na ogół niższa niż kamer monochromatycznych) może okazać się niewystarczająca. Najważniejszy problem wiąże się jednak z tym, iż dla wielu podstawowych operacji przetwarzania obrazów, takich jak: wyrównywanie histogramu szarości (w celu poprawienia kontrastowości obrazu), filtracja, konturowanie, szkieletyzacja itp., istnieją algorytmy jedynie w wersji monochromatycznej. Z pozoru może wydawać się, że algorytmy te można stosować oddzielnie dla każdej z monochromatycznych składowych obrazu barwnego, aby po wykonaniu operacji powrócić do syntezy obrazu barwnego. Jest to z pewnością postępowanie najprostsze, ale nie zawsze właściwe. Ponieważ monochromatyczne składowe obrazu nie są od siebie statystycznie niezależne, można łatwo wykazać, iż na przykład wykonywana w ten sposób filtracja obrazów nie będzie optymalna. W innych przypadkach podobne postępowanie z obrazami barwnymi może powodować powstawanie niepożądanych artefaktów.

Oprogramowanie

Współczesne systemy komputerowego przetwarzania obrazów dla potrzeb diagnostyki lekarskiej lub badań biologicznych są budowane jako systemy dialogowe o rozwiniętych środkach komunikacji z użytkownikiem. Pod tym względem różnią się one zatem od systemów wizyjnych tworzonych – na przykład – dla potrzeb robotyki. Komunikacja z systemem odbywa się za pomocą poleceń języka manipulacji obrazami, wprowadzanych z klawiatury systemu, a dodatkowo środków bezpośredniej komunikacji graficznej, tj. myszy, pióra świetlnego itp., umożliwiających zwłaszcza wprowadzanie parametrów procedur (współrzędnych punktów, konturów itp.) oraz wybór opcjonalnych procedur przetwarzania podpowiadanych przez menu. Podstawowe oprogramowanie takich systemów jest ułatwione dzięki temu, iż w końcu lat siedemdziesiątych podjęto prace nad międzynarodowym standardem oprogramowania tzw. jądra systemu graficznego, znanym dziś pod skrótem GKS (ang. *Graphical Kernel System*) [2]. Takich standardów nie ma jednak dla wyższych poziomów oprogramowania. Nie jest to nic zaskakującego, ponieważ firmy oferujące systemy użytkowe w wielu przypadkach dostarczają je z oprogramowaniem zamkniętym, stanowiącym ich prawnie chronioną własność, nawet jeśli zastosowane w nich algorytmy są powszechnie znane z literatury. Koszt oprogramowania użytkowego systemu niekiedy znacznie przekracza koszt sprzętu wraz ze standardowym oprogramowaniem podstawowym. Względy ekonomiczne przemawiają zatem za tworzeniem oprogramowania użytkowego systemów przeznaczonych do najbardziej typowych zastosowań siłami zespołów programistów w kraju, zamiast zakupu takiego oprogramowania za granicą. Nie dotyczy to jednak oprogramowania systemów komputerowej analizy obrazów stanowiących integralną część większych urządzeń diagnostycznych, na przykład tomografów komputerowych.



Rys. 3. Półautomatyczny system analizy obrazów konturowych, CPO-3/IBM PC XT przeznaczony do badań patomorfologicznych



Rys. 4. IMPAS (*Interactive Modular Picture Analyzing System*) – system analizy obrazów ogólnego przeznaczenia, zawierający procesor przepływowy

W ostatnich latach prace nad komputerowymi systemami analizy obrazów prowadzono w Polsce w kilku ośrodkach. Można wśród nich wymienić zwłaszcza Akademię Górniczo-Hutniczą w Krakowie, Politechnikę Łódzką, Politechnikę Warszawską, Zakład Systemów Automatyki Kompleksowej PAN w Gliwicach oraz Instytut Biocybernetyki i Inżynierii Biomedycznej PAN w Warszawie. W nieco innym aspekcie, bardziej związanym z rekonstrukcją obrazów, podobne prace prowadzono w Instytucie Podstaw Informatyki PAN w Warszawie. Prace te zaowocowały licznymi udanymi rozwiązaniami systemowymi o różnorodnym przeznaczeniu. Przykładowo można tu wymienić opracowane w IBiIB PAN systemy:

- VIST, współpracujący z jednostką centralną mikrokomputera typu IBM PC/AT,
- IMPAS, zawierający autonomiczny, wysoko wydajny procesor przepływowy do przetwarzania obrazów,
- CPO-3/IBM PC/XT, półautomatyczny system do analizy morfometrycznej obrazów konturowych,
- CPO/M68010, system przystosowany zwłaszcza do badań angiograficznych metodą różnicową.

Bardziej szczegółowe dane o tych systemach można znaleźć w pracach [1, 6]. Badania medyczne za pomocą komputerowych systemów analizy obrazów są prowadzone w wielu ośrodkach, m.in. w Centrum Medycznym Kształcenia Podyplomowego w Warszawie (badanie mechanizmów lokomocyjnych komórek), w Akademii Medycznej w Warszawie (rozpoznawanie wczesnych stadiów raka pęcherza moczowego), w Akademii Medycznej we Wrocławiu (badania cytogenetyczne). Znacznie mniej optymistycznie – w stosunku do realnych potrzeb – przedstawia się jednak lista ośrodków służby zdrowia, w których komputerowe systemy analizy obrazów są wykorzystywane w procesach leczenia. Lista ta obejmuje jedynie kilka klinik lub większych szpitali. Jest to, oczywiście, związane z ogólną sytuacją materialną służby zdrowia w Polsce. Nie jedna to zresztą dziedzina, w której możliwości informatyki nie są jeszcze w pełni wykorzystane.

LITERATURA

[1] Dulewicz R.: Zastosowanie optycznej transformacji Fouriera w optoelektronicznym hybrydowym systemie w celu analizy obrazów biomedycznych. Biuletyn Techniczno-Informacyjny MERA, nr 5-6, 1989

[2] Enderle G., Kansy K., Pfaff G.: Computer Graphics Programming. GKS – The Graphics Standard. Springer Verlag, Berlin, 1984

[3] Giloi W. K.: Interactive Computer Graphics. Prentice Hall, 1978

[4] Kulikowski J. L.: Cybernetyczne układy rozpoznające. PWN, Warszawa, 1972

[5] Kulikowski J. L.: Kierunki rozwojowe i metody automatycznej klasyfikacji. INFORMATYKA, nr 4, 5, 1974

[6] Kulikowski J. L.: Systemy komputerowego przetwarzania obrazów dla potrzeb diagnostyki lekarskiej. Biuletyn Techniczno-Informacyjny MERA, nr 5-6, 1989

[7] Rosenblatt F.: The perceptron, a probabilistic model for information storage and organization. Psychol. Rev., No. 65, 1958.

Bardzo prosimy Autorów przysyłających teksty do opublikowania w INFORMATYCE o podawanie numeru telefonu (umożliwiającego bezpośredni kontakt w godzinach pracy) oraz adresu domowego i ewentualnie numeru konta bankowego.

Micomp

ZAKŁAD SYSTEMÓW MIKROKOMPUTEROWYCH

MiCOMP

40–045 Katowice, ul. Astrów 7
telefon 513-086, telefaks 518-628,
teleks 315687

Systemy teletransmisji danych ICL/ODRA

Integracja systemów ICL i ODRA z mikrokomputerami standardu PC i systemami NOVELL, UNIX SCO

procesor sieci rozległej WAN
MiCOMP 8075
(emulacja ICL 7503)

most NOVELL – ICL/ODRA
NetCROSS-FXBM

program teletransmisji danych **MiCROS-FXBM** (PC, rozproszone bazy danych, wersja sieciowa – NOVELL), skaner MPXSCAN-8007,

adaptery, modemy, testery, emulatory terminali.

Instalacje w największych systemach na terenie całego kraju!

E0/1124/89

UWAGA UŻYTKOWNICY SYSTEMÓW KOMPUTEROWYCH PRACUJĄCYCH POD KONTROLĄ S.O. UNIX!

PRZEDSIĘBIORSTWO TECHNICZNO-HANDLOWE
„KaNet" Sp. z o.o.
41–500 CHORZÓW, ul. Katowicka 77
tel. 412-122, teleks 312627 kanet pl

oferuje:

■ Oprogramowanie narzędziowe pakietu autorskiego mixPACK:

☆ **mixCOMM** – **program wspomagający zarządzanie plikami (wyróżnienie oraz nagroda Ministra Przemysłu na Ogólnopolskich Targach Oprogramowania SOFTARG '90):**
rozwinięcie koncepcji programu NORTON COMMANDER, umożliwiające pracę w środowisku wielodostępnym lub sieci lokalnej (s.o. UNIX, XENIX i pochodne);

☆ **mixEDIT** – **pełnoekranowy edytor tekstowy:**
wzorowany na programie NORTON EDITOR, przeznaczony m.in. do pracy w środowisku wielodostępnym lub sieci lokalnej (s.o. UNIX, XENIX i pochodne, MS DOS), ukierunkowany na prace programistyczne;

☆ **mixOFFI** – **pakiet wspomagający prace biurowe:**
implementacja koncepcji pakietów zintegrowanych dla systemów operacyjnych UNIX System V lub pochodnych (w tym XENIX 286 i 386 firmy SCO) oraz systemu DOS;
uniwersalny kalendarz wraz z terminarzem, arkusz kalkulacyjny, baza danych o parametrach definiowanych przez użytkownika, tabela kodów ASCII.

■ Oprogramowanie aplikacyjne z zakresu organizacji i zarządzania przedsiębiorstwem,

■ Nowoczesne rozwiązania sprzętowe firmy Texas Instruments w dowolnych konfiguracjach.

POSIADAMY PRAWA AUTORSKIE NA CAŁOŚĆ OFEROWANEGO OPROGRAMOWANIA!

GWARANTUJEMY WIELOLETNIE UŻYTKOWANIE ZAKUPIONEGO OPROGRAMOWANIA DZIĘKI MOŻLIWOŚCI PRZENOSZENIA GO NA DOWOLNE SYSTEMY KOMPUTEROWE!

ZAPRASZAMY DO KORZYSTANIA Z NASZEJ OFERTY!

0/16/90

WOJCIECH SKABA
Toruń

Współczesne metody kompresji danych: teoria, algorytmy i zastosowania

Efektywne algorytmy kompresji danych są znane od lat pięćdziesiątych, jednak dopiero od niedawna wchodzą w obszar praktycznych zastosowań. Rozpowszechniły się one głównie w dziedzinie przechowywania informacji i transmisji danych, choć można wskazać również inne, często nieoczekiwane przykłady zastosowań. Podstawową klasę stanowią tutaj algorytmy kompresji bez utraty informacji (ang. *lossless compression*) i one będą głównym tematem tego artykułu.

Co to jest kompresja danych?

Zbiorem źródłowym (ang. *source*) nazywa się dowolny ciąg znaków S z pewnego alfabetu $\Sigma = \{a_0, a_1, ..., a_N\}$. Przykładem alfabetu może być zbiór {0, 1}, czyli tzw. alfabet dwójkowy lub zbiór 8-bitowych kodów ASCII { 00, ..., FF}. Szczególnym przykładem zbioru źródłowego jest plik zapisany bajtowo na dysku.

Kompresją danych nazywa się proces przetwarzania pewnego zbioru źródłowego S w zbiór wynikowy (ang. *destination*) D mniejszy od S, w taki sposób, że zbiór S można odtworzyć na podstawie informacji zapisanej w zbiorze D. Konwersję $S \rightarrow D$ nazywa się kompresją lub kodowaniem (ang. *encoding*), a proces odwrotny $D \rightarrow S$ – dekompresją lub dekodowaniem (ang. *decoding*). Powyższą definicją podstawową będzie można później nieco zmodyfikować, poczynając od następujących trzech uwag:

1. Dekompresja wiąże się z pracą pewnego programu dekodującego, który zawiera w sobie część informacji potrzebnej do odtworzenia zbioru S na podstawie D. Pewne definicje dodają do zbioru D program dekodujący mówiąc, że D jest programem generującym zbiór S, zawierającym w sobie program właściwy i dane. Jednak taka definicja uzależnia efektywność algorytmu kodującego od języka maszynowego konkretnego komputera i implementacji. Ponadto, w większości zastosowań, zwłaszcza gdy program dekodujący ma charakter uniwersalny oraz liczba danych jest duża, wielkość programu dekodującego staje się pomijalna. Podobnie, jeżeli mówimy o transmisji danych, to program dekodujący znajduje się u odbiorcy i nie jest przesyłany.

2. Algorytmy kodujące zachowują się różnie przy różnych zestawach danych. Swego czasu, gdy zaczęto przesyłać zagęszczone już pliki za pomocą modemów dokonujących kompresji w trybie on-line, okazało się nawet, że liczba przesyłanych bitów nie maleje, lecz wzrasta! Dlatego warunek, że D jest mniejszy od S, należy traktować bardziej jako intencję twórcy, niż jako warunek uznania programu za realizujący kompresję.

Nie każde dane można poddać kompresji. W praktyce jednak każdy zestaw danych ma pewną nadmiarowość, powtórzenia lub prawidłowości, które będą podstawą późniejszej kompresji. Owe potencjalne możliwości zbiorów źródłowych postaramy się później opisać za pomocą pojęcia entropii.

3. Wyróżnia się dwie zasadnicze klasy algorytmów do kompresji danych:
- algorytmy typu *lossless compression*, w których proces kompresji (dekompresji) odbywa się bez utraty informacji, czyli że można odtworzyć dokładnie oryginalny zbiór S,
- algorytmy typu *lossy compression*, tj. algorytmy w których wystarcza jedynie możliwość odtworzenia pewnego przybliżenia zbioru S.

Algorytmy typu *lossy* mają zastosowanie przy kompresji obrazów, gdzie kosztem utraty dokładności uzyskuje się nieporównanie większe stopnie kompresji. W artykule omawia się jednak wyłącznie bardziej rozpowszechnione algorytmy typu *lossless*.

Zastosowania

Przed omówieniem metod warto w kilku słowach wspomnieć o zastosowaniach, aby wyrobić sobie pewne pojęcie o oczekiwaniach w stosunku do tego rodzaju algorytmów.

Kompresję stosuje się głównie przy przechowywaniu i transmisji danych. Najczęściej danymi są obrazy, teksty lub dane dowolne. Dla tych ostatnich rozpowszechniły się na rynku uniwersalne programy archiwujące [8], takie jak ZOO, ARC, PKARC, LHARC, PAK, PKZIP, QBIT (wymieniono w kolejności od najsłabszego do najlepszego).

Do kompresji obrazów stosuje się zarówno algorytmy typu *lossless*, jak i *lossy*. Stanowi to obecnie poważny problem, gdyż wiele firm chce przesyłać duże (kilka megabajtów) obrazy za pomocą modemów, a dotychczasowe metody z trudem zmniejszają ich rozmiary o połowę. W pracy [1] omówiono bardzo zaawansowane, wykorzystujące fraktale, podejście do problemu *lossy compression*, które pozwala osiągnąć bardzo duże stopnie kompresji, nawet rzędu 10 000 razy.

Do kompresji tekstów stosuje się algorytmy specjalizowane, zbliżone do algorytmów uniwersalnych. Obecnie nieodłączną częścią każdego oprogramowania jest wbudowany system podpowiedzi HELP, a perspektywa 5...6-krotnego zmniejszenia rozmiaru bazy wiedzy jest nie do pogardzenia ani przez producenta oprogramowania, ani przez użytkownika. Jak jednak zobaczymy później, różnica w podejściu wynika nie tylko z odmienności charakteru tych danych.

Dwutygodnik *Electronic World News*, z 7 maja 1990, podał informację o wyprodukowaniu przez firmę InfoChip Systems Inc. z Kalifornii układu IC-105 dokonującego sprzętowej realizacji algorytmu zbliżonego do LZW. Główną jego zaletą jest duża szybkość działania w porównaniu z realizacjami programowymi: kompresja 2 MB/s, dekompresja 5 MB/s, przy zachęcającej cenie 80 dolarów. Procesory takie mają zastosowanie w modemach do transmisji danych, w sterownikach dysków stałych, a w wariancie *lossy*, do szybkiego kodowania obrazów.

Kryteria oceny algorytmów

Zasadniczym celem kompresji danych jest minimalizacja zbioru źródłowego. Nie zawsze jednak daje się to pogodzić z innymi wymaganiami. Warto zatem wymienić po kolei kryteria, które będą służyły do oceny przydatności algorytmów:

Stopień kompresji. Będzie oceniany w procentach jako stosunek rozmiarów D do S, czyli $k = D/S * 100\%$. Niektórzy autorzy podają również współczynnik $k' = 100\% - k$ lub stosunek S/D – krotność kompresji.

Czas kompresji. Jest regułą, że programy osiągające większe stopnie kompresji działają dłużej. Złożoność obliczeniowa wielu algorytmów jest liniowa (np. LZW), chociaż niektóre (głównie te osiagające lepsze wyniki) mają większą złożoność, często niewielomianową.

Czas dekompresji. Celowo wyróżniono tu osobno czas dekompresji, gdyż może on (np. w LZW), choć nie musi (np. w algorytmie Huffmana), być porównywalny z czasem kompresji. Trzeba również dodać, że o ile czas kompresji odnosi się zazwyczaj do całego zbioru źródłowego, to dekompresji poddaje się często jedynie niewielkie fragmenty większej całości. Wówczas, oprócz tempa dekompresji (mierzonego np. w kilobajtach na sekundę) istotny będzie również „czas martwy" dostępu do pierwszego bajtu.

Wymagana pamięć RAM (głównie przy kompresji). Istotnym kryterium jest rozmiar obszaru roboczego, którego ewentualna wirtualizacja na dysku jest niewskazana ze względów czasowych. Interesujące jest przy tym zarówno wielkość pamięci RAM, jak i to, czy zależy ona od wielkości zbioru źródłowego. Niestety, najlepsze algorytmy kompresji wymagają obszaru RAM porównywalnego ze zbiorem źródłowym. Kryterium pamięciowe posłuży do zdefiniowania algorytmów on-line.

Przejrzystość (ang. *clarity*) **zbioru skompresowanego.** Jest określona przez zdolność programu dekodującego do niezależnego odtwarzania możliwie małych fragmentów kodu, a nie np. tylko całego zbioru jednocześnie. Przykładowo, systemy typu hypertext i on-line help wymagają szybkiego dostępu do małych, powiązanych ze sobą modułów, podczas gdy programy archiwujące starają się operować wyłącznie całym zbiorem. Można udowodnić, że algorytmy gorzej spełniające kryterium przejrzystości osiągają większe stopnie kompresji.

Definicja algorytmu typu on-line

Algorytm kompresji (dekompresji) można uznać za typu on-line, jeżeli obszar roboczy algorytmu (RAM) jest stały – niezależny od wielkości zbioru. Definicję tę należy rozumieć w ten sposób, że odczyt zbioru źródłowego i zapis do zbioru wynikowego mogą być dokonywane wyłącznie sekwencyjnie (jednokrotnie), a obszar roboczy RAM (nazywany czasem lokalnym słownikiem), stanowi swego rodzaju strefę buforową. Często obszar RAM algorytmu typu on-line jest większy od programu realizującego ten algorytm.

Entropia zbioru źródłowego

Poprawne zdefiniowanie entropii umożliwia określenie zawartości informacji zbioru źródłowego, a tym samym, potencjalnych możliwości jego kompresji. Przykładowo, zbiór, w którym wszystkie znaki są równe, nie zawiera żadnej informacji (do zakodowania jednego znaku potrzeba 0 bitów). Z kolei zbiór o wartościach losowych zawiera maksimum informacji. Każdy znak wnosi tyle bitów informacji, ile ich sam zawiera.

Entropię zdefiniujemy dla źródeł pierwszego rzędu. Źródłem pierwszego rzędu (ang. *first-order source*), dla alfabetu:

$$\Sigma = \{a_1, ..., a_k\}$$

nazywa się taki zbiór S, w którym poszczególne znaki są od siebie niezależne. Mówiąc inaczej są dane prawdopodobieństwa $p_1 ... p_k$ wystąpienia poszczególnych znaków alfabetu Σ i nic więcej. Nie ma zatem żadnej informacji o wzajemnej relacji poszczególnych znaków. Entropię definiuje się następująco:

$$H(S) = \sum_i p_i * \log_2(1/p_i)$$

Fundamentalne twierdzenie teorii informacji mówi, że do zakodowania zbioru S potrzeba średnio nie mniej niż $H(S)$ bitów na jeden znak zbioru S. Ponadto, dla każdego $\varepsilon > 0$ istnieje algorytm, który koduje zbiór S z użyciem średnio $H(S)+\varepsilon$ bitów na znak (dla dostatecznie dużego zbioru S). Wynikające stąd następujące fakty są intuicyjnie oczywiste:

- nie można skompresować zbioru o wartościach losowych typu *random*;
- po osiągnięciu entropii, zbiór nie poddaje się już dalszej kompresji;
- nie można zagwarantować, że dany algorytm osiągnie jakikolwiek stopień kompresji na dowolnych danych.

W praktyce rzadko spotyka się źródła pierwszego rzędu. Przykładowo, jest mało prawdopodobne, aby po sekwencji **kompute** następna litera była inna niż *r*. Ta obserwacja prowadzi do pojęcia źródła *i*-tego rzędu, w którym dany znak zależy od *i*-1 poprzedzających go. Modelowanie źródeł wyższych rzędów i wyprowadzanie stąd pojęcia entropii jest w praktyce mało przydatne, jednak wiedza o porządku znaków w zbiorze źródłowym może być z powodzeniem wykorzystana do osiągnięcia większych stopni kompresji. Rozważania teoretyczne pokazują, iż tekst angielski ma entropię rzędu 1-2 bitów na znak.

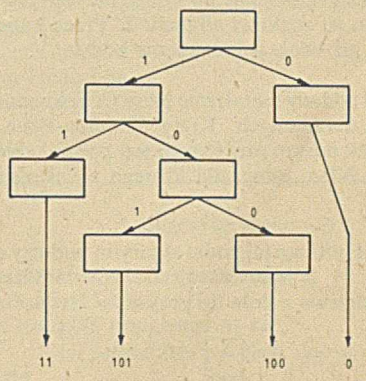
Kody przedrostkowe

Kody przedrostkowe (ang. *prefix codes)* umożliwiają zapis informacji w postaci ciągów bitowych o zmiennej długości i ich jednoznaczną interpretację przy odczytywaniu. Intencją twórców było kodowanie częściej występujących kodów w mniejszej liczbie bitów, kosztem zwiększenia długości kodów występujących rzadziej. Długość kodów przedrostkowych (w bitach) jest określana na podstawie kolejno pobieranych bitów.

Następujący zestaw jest przykładem kodów niejednoznacznie dekodowanych:

1. 0 0
2. 0 1
3. 0 11
4. 1

Chociaż kody 2. i 3. są różne, nie można ich odróżnić na podstawie pierwszych bitów, gdyż nie wiadomo, czy jeżeli po sekwencji 01 trzeci bit będzie jedynką, to oznacza to kod 3., czy kody 2. i 4.



Rys. 1. Drzewo binarne i kody przedrostkowe

Kody przedrostkowe można przedstawić jako liście pewnego drzewa binarnego (rys. 1). Każdy kod rozpoczyna się w miejscu korzenia i składa się z zer i jedynek napotkanych na drodze do odpowiadającego mu liścia. Ruch w lewo oznacza 0, w prawo 1.

Drzewo Huffmana

Abstrahując od przedstawionych wywodów teoretycznych, postawmy jeszcze raz pytanie, dlaczego w ogóle kompresja danych jest możliwa.

Niech będzie dany pewien zbiór S złożony z N 8-bitowych znaków z k-elementowego alfabetu $\Sigma = \{a_1, ..., a_k\}$, $k = 256$. W typowym przypadku jest mało prawdopodobne, aby krotności wystąpień poszczególnych znaków były sobie równe. Można zatem przypisać zbiorowi S zbiór wag $\{w_1, ..., w_k\}$, przy czym w_i oznacza liczbę wystąpień i-tego znaku. Następnie poszczególne znaki zbioru S można zastąpić ich odpowiednikami w postaci niepowtarzalnych kodów bitowych $\{c_1, ..., c_k\}$ różnej długości, przypisanych poszczególnym znakom alfabetu Σ, stosując przy tym zasadę, że znaki z większymi wagami otrzymują mniejszą (lub równą) liczbę bitów. Jest oczywiste, że muszą być to kody przedrostkowe, reprezentowane za pomocą pewnego drzewa binarnego T.

Niech teraz zbiór S składa się w połowie ze znaku a_1 (np. spacji), a w połowie z pozostałych znaków alfabetu Σ. Zbiór wag można zatem przedstawić jako $\{N/2, w_2, ..., w_k\}$. Zastosujmy następujący sposób kodowania: dla znaku a_1 1-bitowy kod 0, dla pozostałych znaków przedrostek w postaci 1-bitowego kodu 1 oraz następującego po nim oryginalnego 8-bitowego kodu znaku. Widać, że tak zakodowany zbiór można zdekodować w sposób jednoznaczny. Ponadto rozmiar bitowy oryginalnego zbioru S wynosi $8*N$, podczas gdy zbiór zakodowany ma:

$$1*N/2 + (1+8)*N/2 = 5*N \text{ bitów} < 8*N \text{ bitów}$$

Powyższym przykładem zilustrowano nietrywialny przypadek, kiedy można osiągnąć dość znaczny stopień kompresji. Wydaje się, że na ogół będziemy w stanie przypisać poszczególnym wagom odpowiednie kody bitowe, tak aby rozmiar zbioru wynikowego był mniejszy od źródłowego. Pojawia się zatem zasadnicze pytanie, w jaki sposób znaleźć drzewo optymalne.

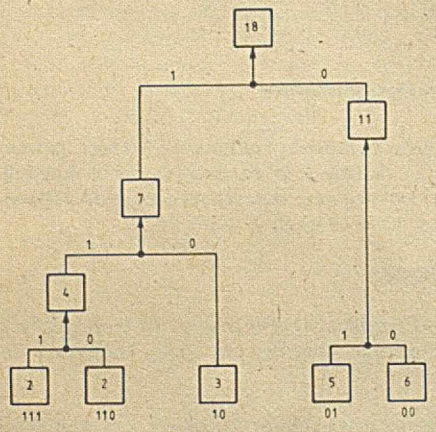
Jeżeli $|c_i|$ oznacza długość (w bitach) kodu c_i, to całkowity rozmiar kodu wynikowego, zakodowanego za pomocą drzewa binarnego T, można zapisać wzorem:

$$W(T) = \sum_i w_i * |c_i|$$

Naszym celem jest minimalizacja $W(T)$. Intuicyjnie nasuwa się następujący algorytm konstrukcji optymalnego drzewa binarnego. Dzieli się zbiór wag $W = \{w_1, ..., w_k\}$ na dwa zbiory W_0 i W_1 w taki sposób, że sumaryczne wagi zbiorów W_0 i W_1 są możliwie najbliższe sobie, to znaczy ich różnica jest jak najmniejsza. Podział taki nie zawsze jest jednoznaczny. Następnie tworzy się 1-bitowe początki kodów c_i dla poszczególnych znaków alfabetu Σ, dając 0 kodom, których wagi znalazły się w W_0, oraz 1 tym, których wagi znalazły się w W_1. Dalej stosuje się algorytm rekurencyjnie, dzieląc zbiory wag na równe części i dopisując 0 lub 1. Jeżeli w kolejnym kroku zbiór wag składa się z jednej tylko wagi, to kończy się rozwijanie tej gałęzi, przypisując utworzony kod odpowiadającemu jej znakowi alfabetu Σ. Pracę kończymy, gdy wszystkie znaki mają już swoje jednoznaczne kody.

Algorytm ten został podany niezależnie przez C. E. Shannona i R. M. Fano w końcu lat czterdziestych. Kody Shannon-Fano są bliskie optymalnym [3], jednak dużym problemem jest podział zbioru wag na dwie części (problem NP-zupełny, dla którego nie istnieje algorytm wielomianowy).

W roku 1952 D. A. Huffman [4] podał algorytm budowy optymalnego drzewa binarnego. W przeciwieństwie do drzewa Shannon-Fano, algorytm ten buduje drzewo z dołu do góry (rys. 2).



Rys. 2. Drzewo Huffmana

Najpierw każdemu znakowi alfabetu a_i przypisuje się jednowęzłowe drzewo T_i z wagą w_i (drzewo składające się z jednego elementu). Następnie rozpoczyna się następującą procedurę rekurencyjną: szuka się dwóch drzew z najmniejszymi wagami, tworzy nowe drzewo z połączenia tych dwóch drzew, z wagą równą sumie wag tych drzew, po czym dodaje nowe drzewo do zbioru drzew i usuwa drzewa składowe. Postępuje się tak długo aż pozostanie tylko jedno ostateczne drzewo.

Intuicyjnie optymalność tego algorytmu wydaje się dość oczywista, gdyż znaki z mniejszymi wagami otrzymują zawsze dłuższe kody. Jeżeli długość zbioru źródłowego zmierza do nieskończoności, to algorytm Huffmana osiąga entropię dla źródeł pierwszego rzędu. Tak więc, jeżeli mamy jedynie informację o wagach poszczególnych znaków, to nie istnieje lepsza metoda kompresji danych niż drzewo Huffmana.

Konstruowanie drzewa Huffmana

Pokażemy teraz efektywną metodę konstruowania drzewa Huffmana. Zacznijmy od ciekawej obserwacji.

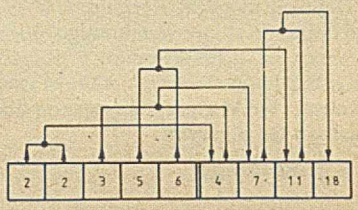
W przedstawionym algorytmie nie sprecyzowano sposobu przypisania dwóch drzew składowych do lewej i prawej gałęzi nowo tworzonego drzewa. Podobnie, w zbudowanym już drzewie, można zamienić miejscami dowolne dwie gałęzie (wraz ze wszystkimi ich pochodnymi) bez uszczerbku dla warunku optymalności. Stąd wniosek, że dzięki temu można posortować wszystkie znaki alfabetu Σ według ich wag (czyli w rezultacie według długości kodów Huffmana). Pozwala to na bardzo oszczędny zapis struktury samego drzewa.

Druga obserwacja warta odnotowania, to fakt, że pierwsze nowo tworzone drzewa, jako mające najmniejsze wagi, są w śród nich zarazem pierwszymi kandydatami do utworzenia kolejnych drzew. W efekcie można skonstruować prosty i szybki algorytm budowania drzewa Huffmana.

Tworzy się tablicę $2*k-1$ elementową: k komórek dla liści – znaków alfabetu i ich wag, $k-1$ dla nowo tworzonych drzew, tj. węzłów ostatecznego drzewa, którego korzeniem będzie komórka o indeksie $2*k-1$ (rys. 3). Niech znaki alfabetu Σ z pierwszych k komórek będą posortowane według zwiększającej się wagi. Kolejno tworzone drzewa będą umieszczane w kolejnych komórkach tablicy, począwszy od komórki o indeksie $k+1$ (ich wagi będą automatycznie posortowane). Algorytm wyboru dwóch kandydatów na nowe drzewo może ograniczyć się do rozważenia jedynie trzech następujących par (o ile istnieją):

- dwa liście o najmniejszych wagach,
- dwa węzły wewnętrzne o najmniejszych wagach,
- liść i węzeł wewnętrzny o najmniejszej wadze.

Jak łatwo zauważyć, położenie tych kandydatów jest w każdej chwili dobrze znane.



Rys. 3. Konstruowanie drzewa Huffmana

Złożoność obliczeniowa tego algorytmu jest liniowa (względem k). Głównym ograniczeniem czasowym jest więc czas sortowania tablicy k-elementowej, rzędu $k*\log(k)$.

Algorytmy dynamiczne i adaptacyjne

Pojęcie algorytmu dynamicznego lub adaptacyjnego nie ma ścisłej definicji. Przedstawiony algorytm kompresji danych za pomocą drzewa Huffmana jest przykładem algorytmu statycznego. Algorytmy dynamiczne dodają pewną zmienność istotnego elementu struktury danych programu dokonującego kompresji w trakcie przetwarzania zbioru źródłowego. Podobnie jest w trakcie dekompresji. Algorytm adaptacyjny jest rodzajem dynamiczności, która oznacza jednocześnie adaptację do aktualnych danych. Pojęcia te nie stosują się do źródeł pierwszego rzędu.

Dynamiczne drzewo Huffmana

Klasyczny algorytm kompresji danych z wykorzystaniem drzewa Huffmana wymaga co najmniej dwukrotnego dostępu do zbioru źródłowego. Pierwszy odczyt służy do zbudowania drzewa Huffmana, drugi – do dokonania właściwego kodowania. Nie spełnia on zatem warunku algorytmu typu on-line. Interesujące byłoby skonstruowanie algorytmu budującego na bieżące drzewo Huffmana wraz z każdym nowo odczytanym znakiem. Algorytm taki pracowałby w trybie on-line, nie wymagałby zapisywania drzewa Huffmana, a jednocześnie miałby pewne cechy algorytmu dynamicznego. Ważne jest, aby drzewo modyfikować po (a nie przed) zapisaniem do zbioru wynikowego kodu danego znaku, gdyż inaczej nie byłoby możliwości rozkodowania zbioru wynikowego.

Własność rodzeństwa (ang. *sibling property*) [2] odnosi się do drzew binarnych, w których waga każdego wewnętrznego węzła jest sumą wag jego dzieci. Drzewo binarne ma własność rodzeństwa, jeżeli są spełnione następujące dwa warunki:

1. Każdy węzeł (liść lub węzeł wewnętrzny) z wyjątkiem korzenia ma odpowiadające mu rodzeństwo.
2. Wszystkie węzły można posortować według niemalejącej wagi w taki sposób, że każdy węzeł jest obok swojego rodzeństwa.

Następujące ciekawe twierdzenie podajemy bez dowodu: Drzewo binarne jest drzewem Huffmana wtedy i tylko wtedy, gdy spełnia ono własność rodzeństwa.

Stąd wniosek, że zamiast budować drzewo Huffmana, wystarczy zbudować drzewo, które spełnia własność rodzeństwa. W konsekwencji, algorytm uaktualniania drzewa Huffmana jest prosty i szybki: pobiera się kolejny znak i rozpoczyna „wędrówkę" w górę drzewa od odpowiadającego mu liścia aż do korzenia (wzdłuż gałęzi). Po drodze zwiększa się wagę każdego napotkanego węzła o jeden (poczynając od liścia) i sprawdza, czy jego waga jest większa od bieżącego sąsiada (tego od strony korzenia). Jeżeli tak, to zamienia się bieżący węzeł z węzłem najbliższym korzeniowi, który ma wagę mniejszą od wagi bieżącego węzła. „Zamiana" oznacza wymianę rodziców przy pozostawieniu dzieci.

Wiedząc już, w jaki sposób szybko przebudować drzewo Huffmana można skonstruować następujący algorytm kodujący: pobiera się kolejny znak i szuka odpowiadającego mu liścia. Jeżeli znak ten był już wcześniej pobierany, to można znaleźć jego liść i odpowiadający mu kod Huffmana, a następnie przebudować drzewo zgodnie z algorytmem omówionym powyżej. Jeżeli natomiast znak ten występuje po raz pierwszy, to trzeba posłużyć się koncepcją **niewidocznego liścia** (ang. *unseen leaf*). Niewidoczny liść nie ma przypisanego żadnego znaku, a jego waga wynosi 0. W stanie początkowym drzewo składa się z samego niewidocznego liścia. Gdy pojawia się nieznany dotychczas znak, to zamiast odpowiedniego kodu Huffmana używa się kodu Huffmana niewidocznego liścia wraz z następującym po nim oryginalnym kodem nowego znaku. Następnie tworzy się z niewidocznego liścia nowe drzewo, zmieniając ten liść na węzeł wewnętrzny i dodając do węzła nowy niewidoczny liść o wadze 0 oraz liść nowego znaku o wadze 1. Ostatnim etapem będzie również uaktualnienie drzewa z wykorzystaniem własności rodzeństwa. Algorytm dekompresji jest analogiczny.

Uwagi na temat drzew Huffmana

Warto podsumować dotychczasową wiedzę dotyczącą drzew Huffmana. Metoda Huffmana jest nie tyle algorytmem kompresji danych, co pewną procedurą optymalizacyjną, która wskazuje najefektywniejsze drzewo binarne dla pewnego zbioru znaków z wagami. Nie jest przy tym istotne, czy są to znaki alfabetu łacińskiego, czy też, np. wszystkie niepowtarzalne wyrazy w tekście (których może być nawet ponad 10 tysięcy). W bardziej zaawansowanym podejściu problem optymalnej kompresji zbioru źródłowego sprowadzi się do problemu wyboru optymalnego alfabetu.

Podstawową wadą metody Huffmana jest to, iż w obecnym ujęciu stosuje się ją z powodzeniem jedynie do źródeł pierwszego rzędu, tzn. nie wykorzystuje żadnej informacji o wzajemnym porządku znaków. Można obejść ten problem stosując odpowiednią heurystykę.

Obecnie warto jednak zająć się algorytmami, które stosują się wyłącznie do kompresji danych, a jednocześnie w sposób naturalny wykorzystują własności źródeł wszystkich rzędów.

Metody podstawień tekstowych

Metody podstawień tekstowych (ang. *textual substitution*) stanowią, obok drzew Huffmana, drugą obszerną klasę algorytmów kompresji danych. Opierają się one na założeniu, że w zbiorze źródłowym znajduje się wiele podobnych do siebie ciągów znaków – łańcuchów – takich, jak np. wyrazy w tekście. Zadaniem programu kodującego jest wyszukanie możliwie długich powtarzających się łańcuchów i zastąpienie (podstawienie) kolejnych fragmentów tekstu wskaźnikami (indeksami) do budowanego jednocześnie słownika wzorów łańcuchowych.

Wyróżniamy tu metody on-line oraz off-line. Metody off-line opierają się na wykorzystaniu dużych mocy obliczeniowych i nieograniczonej pamięci, dzięki czemu można przeprowadzić dość dokładną analizę struktury zbioru źródłowego, a w rezultacie osiągnąć znaczne stopnie kompresji. Nie ma jednak powodu, by omawiać je oddzielnie od np. wzmiankowanego wcześniej problemu poszukiwania optymalnego alfabetu, czy też ogólniej, metod wykorzystujących nieograniczone zasoby komputerowe.

Znacznie ciekawsze są z naszego punktu widzenia metody on-line. Stanowią one bowiem obecnie podstawę większości praktycznych zastosowań algorytmów kompresji danych.

Ogólny model metody on-line

Dotychczasowe rozważania na temat podstawień tekstowych nie wnoszą jeszcze niczego zasadniczo nowego. Można by przecież wyselekcjonować pewien zbiór łańcuchów wzorcowych, po czym potraktować go jako nowy alfabet i zbudować drzewo Huffmana. Istotna nowość pojawia się dopiero w obecnym modelu on-line, a oparcie go na założeniu powtarzalnych ciągów znakowych jest tylko wskazaniem właściwej drogi.

Niech Σ oznacza alfabet, a D lokalny słownik, tj. pewien obszar pamięci, w którym będą przechowywane wzorcowe łańcuchy znaków z alfabetu Σ. Poszczególne łańcuchy będą reprezentowane za pomocą indeksów. Można przyjąć, że Σ składa się z 156 znaków, a D ma pojemność 4096 łańcuchów znakowych (czyli 12-bitowy indeks).

Modelowy algorytm kompresji i dekompresji można teraz zapisać za pomocą *pseudojęzyka*. Obok pojęcia zbioru źródłowego będzie stosowane pojęcie strumienia wejściowego (ang. *input stream*), które lepiej oddaje sekwencyjność dostępu do danych. Ponadto przyjęto następujące oznaczenia:

t,x	łańcuch znaków
X	zbiór łańcuchów
$\{\ \}$	zbiór pusty
D	słownik

Algorytm kompresji

```
(1) Zainicjowanie lokalnego słownika D.
    InitializationHeuristic (D);
(2) while not EOF (inputstream) do
                (* powtarzaj do końca zbioru źródłowego *)
      begin
    (a) Pobranie nowego skojarzenia (match):
        Pobierz nowy łańcuch za pomocą MatchHeuristic
          t := MatchHeuristic (inputstream);
        Prześlij indeks łańcucha t do zbioru wynikowego.
    (b) Uaktualnienie lokalnego słownika D.
        Wyznacz zbiór nowych łańcuchów:
          X := UpdateHeuristic (D);
        Dodaj łańcuchy ze zbioru X do słownika:
          while X <> { } do
            begin
              Pobierz kolejny łańcuch x z X.
              if (nie ma x w D) then
                begin
                  if (D jest pełny) then
                    DeletionHeuristic (D);
                    (* zrób miejsce w D na łańcuch x *)
                  Dodaj x do D.
                end
            end
end (* not EOF *)
```

Wyjaśnienia wymagają nazwy użytych heurystyk.

InitializationHeuristic: inicjowanie słownika D. Aby inicjowanie było poprawne, słownik D musi zawierać co najmniej jednoznakowe łańcuchy alfabetu Σ.

MatchHeuristic: skojarzenie ciągu znaków strumienia wejściowego z bieżącą zawartością słownika D. Pobiera się łańcuch, który najlepiej odpowiada naszym wymaganiom oraz bieżącej zawartości słownika. Może to być np. najdłuższy łańcuch w D pasujący do bieżącego strumienia wejściowego.

UpdateHeuristic: uaktualnienie słownika D, czy inaczej, funkcja określająca zbiór X łańcuchów, które mają być dodane do słownika D, w wyniku znalezienia nowego skojarzenia. UpdateHeuristic współpracuje z DeletionHeuristic, która, w razie potrzeby, tworzy miejsce na nowe łańcuchy.

DeletionHeuristic: skasowanie części łańcuchów w słowniku D w celu uzyskania miejsca na nowe łańcuchy. Funkcja określa, które łańcuchy należy usunąć, aby uszczerbek był jak najmniejszy. Mogą to być np. łańcuchy najrzadziej używane.

A oto omówienie algorytmu modelowego zapisanego w pseudojęzyku. Na początku inicjuje się słownik D. Aby algorytm działał poprawnie na dowolnych danych, słownik musi zawierać przynajmniej łańcuchy

będące jednobajtowymi odpowiednikami znaków alfabetu Σ, których nie wolno usuwać za pomocą DeletionHeuristic. Wówczas, w najgorszym przypadku, zbiór źródłowy zakoduje się jako indeksy łańcuchów jednobajtowych. Faza (2) algorytmu to właściwy program kodujący. Składa się on z dwóch faz: pobrania nowego łańcucha (a), oraz uaktualnienia słownika *D* (b).

Faza (a) jest realizowana za pomocą funkcji MatchHeuristic, której zadaniem jest pobranie ze strumienia wejściowego pewnego ciągu znaków *t*, któremu można przyporządkować pewien łańcuch zawarty w słowniku *D*. W następstwie tego skojarzenia, odpowiadający temu łańcuchowi indeks będzie przesłany do zbioru wynikowego.

Faza uaktualniania słownika (b) jest realizowana naprzemiennie z fazą (a), po wytransmitowaniu każdego nowego indeksu. Zadaniem tej fazy jest takie przetworzenie słownika *D*, by przyszła praca kodera była możliwie najefektywniejsza. UpdateHeuristic określa zbiór łańcuchów, które mają być dodane do słownika. DeletionHeuristic określa, które łańcuchy można usunąć ze słownika, jeżeli brakuje miejsca na nowe.

Po zainicjowaniu słownik jest prawie pusty i rozpoczyna się stopniowe jego zapełnianie. Jest to faza, w której nie ma potrzeby stosowania DeletionHeuristic, a jednocześnie osiągane stopnie kompresji są niewielkie, gdyż algorytm dysponuje małym zasobem wzorców dla MatchHeuristic.

Gdyby zakończyć uaktualnianie słownika po jego zapełnieniu (np. bez stosowania DeletionHeuristic), to byłby to algorytm statyczny. Ściślej: *algorytmem statycznym jest algorytm, który ma tylko fazę inicjowania, chociaż w praktyce faza ta może wykorzystywać Update-* i DeletionHeuristic, a nawet dokonywać kompresji początkowej części zbioru źródłowego. Jego zaletą jest szybkość, wadą natomiast brak jakiegokolwiek stopnia adaptacyjności do później nadchodzących danych. Z kolei, zaletą algorytmu dynamicznego jest ciągła zmienność słownika. Zaznaczmy, iż wówczas faza początkowa jest (intencjonalnie) pomijalna w porównaniu z fazą pracy pełnego algorytmu.

Dla porządku warto jeszcze zapisać w pseudojęzyku modelowy algorytm dekompresji, o analogicznej strukturze:

Algorytm dekompresji

```
(1) Zainicjowanie lokalnego słownika D.
    (* powtórzenie fazy (1) algorytmu kompresji *)
(2) while not EOF (inputstream) do
        (* powtarzaj aż do końca zbioru wynikowego *)
    begin
    (a) Pobranie nowego skojarzenia (match):
        Pobierz nowy indeks.
        Prześlij łańcuch t odpowiadający temu indeksowi.
    (b) Uaktualnienie lokalnego słownika D.
        (* powtórzenie fazy (2b) algorytmu kompresji *)
    end (* not EOF *)
```

Przedstawiony model kompresji metodą podstawień tekstowych on-line nie jest ani jednoznaczny (ze względu na możliwość stosowania różnych heurystyk), ani zamknięty (ze względu na możliwość wprowadzania zmian do modelu podstawowego). Istotna różnica, w stosunku do metody Huffmana, polega na tym, że kody są stałej długości, nieprzerwanie natomiast zmienia się słownik, tak że dany kod może w każdej chwili oznaczać co innego. Okazuje się, że algorytm ten jest optymalny w sensie informacyjno-teoretycznym, tak że prawdopodobieństwo wystąpienia każdego indeksu jest jednakowe i wszelkie pomysły dalszego poprawiania stopnia kompresji za pomocą drzew Huffmana, obliczanych dla wynikowego ciągu indeksów, nie dają żadnej poprawy.

Przedstawiony model wydaje się wręcz idealny z punktu widzenia praktycznych zastosowań: pracuje w trybie on-line, programy są szybkie i nie wymagają dużej pamięci, zbiorem wynikowym jest sam ciąg indeksów (tzn. nie ma słowników, drzew itp.), a w praktycznych realizacjach uzyskuje się znaczne stopnie kompresji, niezależnie od danych wejściowych.

Istnieje wiele praktycznych wariantów realizacji modelowego algorytmu. Oto niektóre z nich.

Alfabetem Σ jest najczęściej zbiór 8-bitowych kodów ASCII { # OO, ..., # FF }. Długość indeksów waha się od 9 bitów (512 łańcuchów) do 13 bitów (8192 łańcuchy). Najbardziej popularną wartością, optymalną dla plików tekstowych, jest 12 bitów. Niektóre rozwiązania dopuszczają zmiany długości indeksu w trakcie pracy algorytmu.

InitializationHeuristic organicza się z reguły do jednobajtowych dopowiedników alfabetu Σ.

MatchHeuristic to, w najprostszym podejściu, najdłuższy łańcuch pasujący do bieżącego strumienia znaków. Bardziej zaawansowane heurystyki dopuszczają rozwiązania, w których „przewiduje się" kilka najbliższych skojarzeń (ang. *lookahead*).

Poniżej przedstawiono trzy popularne rozwiązania dla UpdateHeuristic, określającej jakie łańcuchy mają być dodane do słownika:

FC: (*first character*) – pierwszy znak → nowy łańcuch = bieżące skojarzenie + pierwszy znak następnego skojarzenia.
ID: (*identity*) – tożsamość → nowy łańcuch = bieżące skojarzenie + całe następne skojarzenie.
AP: (*all prefixes*) – wszystkie przedrostki → nowe skojarzenie = = bieżące skojarzenie + wszystkie przedrostki następnego skojarzenia (włączając je same).

Posłużmy się przykładem z publikacji [7]. Niech bieżącym skojarzeniem będzie wyraz „THE_", a następnym „CAT". Wówczas poszczególne heurystyki dadzą następujące zbiory łańcuchów:

FC: { "THE_C" }
ID: { "THE_CAT" }
AP: { "THE_C", "THE_CA", "THE_CAT" }

Warto przyjrzeć się bliżej pracy algorytmu modelowego zaraz po inicjacji, z użyciem heurystyki AP i bez DeletionHeuristic. Niech będzie dany ciąg znaków:
"THE_CAT_AT_CAR_ATE_THE_RAT"

Lewa kolumna będzie zawierać bieżące skojarzenia, a prawą – zbiory łańcuchów dodawanych do słownika.

Skojarzenia	Nowe łańcuchy	Komentarz
T		;skojarzenie dzięki inicjacji, brak nowych łańcuchów
H	TH	
E	HE	
_	E_	;E + _ = E_
C	_C	
A	CA	
T	AT	;nowy łańcuch AT
_	T_	
AT	_A _AT	;pierwsze skojarzenie z nieinicjowanym łańcuchem, dwa nowe łańcuchy
_	AT_	
TH	_T _TH	
E_	THE THE_	
CA	E_C E_CA	
R	CAR	
_AT	R_A R_AT	
E_	_ATE _ATE_	
THE_	E_T E_TH E_THE E_THE_	;aż cztery nowe łańcuchy
R	THE_R	
AT	RAT	

Jako ostatnie omówmy typowe realizacje dla funkcji DeletionHeuristic, określającej które łańcuchy mają być usunięte ze słownika.
FREEZE: zamrożenie → zbiór łańcuchów do usunięcia jest pusty. Jest to najprostsze rozwiązanie, które można określić mianem „statycznego", kończącego uaktualnianie słownika po krótkiej fazie inicjowania.
LRU: (*least recently used*) – najdawniej używany → jest usuwany ten łańcuch, który spośród łańcuchów w słowniku był najdawniej kojarzony ze strumieniem wejściowym. Jest to prosta do realizacji i bardzo efektywna heurystyka.
LFU: (*least frequently used*) – najrzadziej używany → jest usuwany ten łańcuch, który był najrzadziej używany. Dodatkowo funkcja ta musi rozstrzygać konflikty między łańcuchami używanymi tę samą liczbę

razy. Pod względem efektywności LFU jest porównywalne do LRU, ale trudniejsze do implementacji.

SWAP: zamiana → używa się równocześnie dwa słowniki. Gdy pierwszy się zapełni, rozpoczyna się zapełnianie drugiego, ale wykorzystuje się pierwszy. Po zapełnieniu drugiego słownika, kasuje się pierwszy i role słowników zamieniają się. Efektywność jest zbliżona do LRU. Podejście to odbiega nieco od podstawowego modelu.

Algorytm dynamiczny Lempel-Ziv-Welcha

Najbardziej popularną implementacją metody podstawień tekstowych jest algorytm określany mianem LZW. Pochodzi on z pracy Ziva Lempela [10], z pewnymi modyfikacjami pochodzącymi od Welcha [9]. Welch jednak rozważa mało efektywną heurystykę FC-FREEZE. Późniejsze prace, np. Millera i Wegmana [6], opierają się już na heurystyce FC-LRU i ten wariant „dynamiczności" jest najczęściej spotykaną praktyczną realizacją metody LZW.

Prace oryginalne pozostawiają swobodę co do pewnych szczegółów implementacyjnych, jak np. długość indeksu lub rodzaj użytej heurystyki. Ponadto, przejrzystość wzorcowego algorytmu jest z reguły tracona na skutek wprowadzania mechanizmów ukierunkowanych na skrócenie czasu realizacji programu.

Adaptacyjna kompresja Huffmana z heurystyką odświeżanie-zanik

Kolejny przykład ilustruje możliwości wykorzystywania podstawowej wiedzy o metodach kompresji danych do konstruowania bardziej złożonych algorytmów. Algorytm *Adaptive Huffman Squeezing with Refresh-Decay Heuristic*, opracowany przez autora, z powodzeniem stosuje do kompresji pewnych typów danych np. obrazów. Obrazy charakteryzują się tym, iż nie nadają się do kompresji ani metodą LZW (brak powtarzających się ciągów), ani standardową metodą Huffmana (dość równomierny globalny rozkład znaków). Z drugiej jednak strony, występują tam pewne prawidłowości, jak chociażby duże plamy składające się ze zbliżonych kolorów, łagodne przejścia między plamami itp. Nie będziemy tu wchodzić w rozważania na temat, kiedy zastosowanie tego algorytmu może być opłacalne. Skupimy się jedynie na omówieniu podstawowej idei.

Załóżmy że jest dany pewien alfabet Σ zbiór źródłowy S oraz drzewo Huffmana zbudowane dla pewnego podzbioru Σ. Po wczytaniu każdego znaku zapisuje się do zbioru wynikowego odpowiadający mu kod Huffmana, a następnie przebudowuje drzewo. Znaki mogą być dodawane bądź usuwane z drzewa, tak że z reguły tylko niewielka część alfabetu Σ będzie rezydować w drzewie. Jednym z liści drzewa będzie omawiany już wcześniej niewidoczny liść, któremu nie odpowiada żaden znak alfabetu Σ, a który nie może być usunięty. Jego rola będzie identyczna jak w przypadku omawianego dynamicznego drzewa Huffmana, z tym że nadamy mu pewną niezerową wagę.

Każdemu znakowi rezydującemu w drzewie będzie przypisana pewna dodatnia liczba rzeczywista (waga), która w sposób jednoznaczny określa jego lokalizację w drzewie, a wszystkie razem strukturę całego drzewa. Waga będzie zmienna w czasie. Jeżeli waga zmniejszy się poniżej pewnej dolnej granicy, to odpowiadający jej znak zostanie usunięty z drzewa. Granica ta będzie określona przez porównanie bieżącej długości kodu Huffmana z sumaryczną długością kodu niewidocznego liścia i oryginalnego kodu tego znaku.

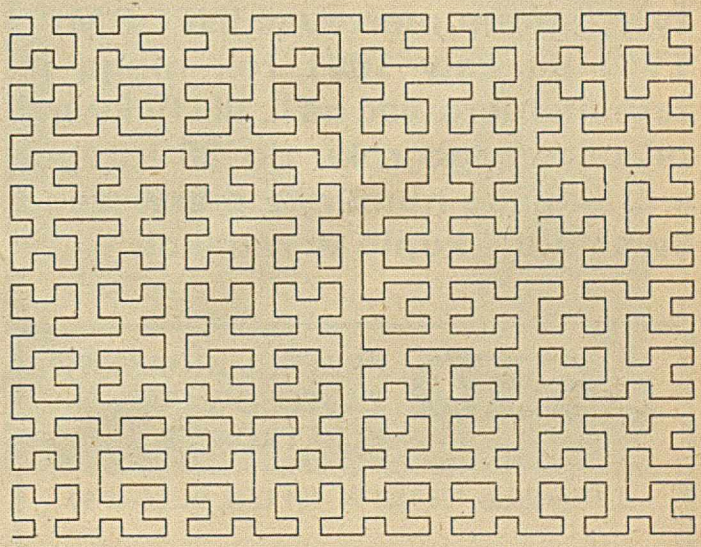
Najistotniejszym elementem tego algorytmu jest heurystyka określająca zasady zmieniania wag poszczególnych znaków. Oczywiście najlepiej byłoby, gdyby znaki same „wiedziały", jakie wagi mają w każdej chwili przyjmować. Jeżeli jest to niemożliwe, to trzeba tę zmienność zasymulować. Heurystyka odświeżanie-zanik (ang. *refresh-decay*) przypomina swoim zachowaniem działanie kondensatora. Istnieje pewna stała dodatnia C, określająca wielkość odświeżania, oraz pewien stały parametr K, nieco mniejszy od 1.0, określający tempo zaniku. Odświeżanie jest liniowe, podczas gdy zanik jest wykładniczy. Parametry C i K są dobierane doświadczalnie. Z każdym krokiem pracy algorytmu (tj. wraz z każdym wczytanym znakiem) wszystkie wagi znaków rezydujących w drzewie są mnożone przez K. Następuje w ten sposób równomierne (w skali logarytmicznej) obniżenie wagi każdego znaku. Jednocześnie waga znaku, który został w tym kroku wczytany, jest zwiększana o C. Jeżeli jest to nowy znak, to jego waga jest inicjowana również na C. Interesujące jest, w jaki sposób zmienia się waga niewidocznego liścia. Otóż przyjęto zasadę, że zmienia się ona tak samo, jak waga każdego innego znaku. Pojawienie się nowego znaku jest równoznaczne z odświeżeniem również niewidocznego liścia. Tak więc z każdym nowym znakiem następuje dwukrotne uaktualnienie wag; najpierw dla niewidocznego liścia, potem dla nowego znaku. Oczywiście, wraz z każdym uaktualnieniem wag drzewo Huffmana jest przebudowywane. Dodajmy jeszcze, że po zainicjowaniu drzewo składa się z samego niewidocznego liścia z wagą równą C.

Interesujące mogą być również szczegóły implementacyjne. Przechowywanie wag w postaci zmiennoprzecinkowej i mnożenie wag przez stałą wraz z każdym wczytanym znakiem byłoby zbyt czasochłonne. Trzeba by również wówczas od nowa budować drzewo Huffmana dla zupełnie nowych wag. Dlatego wagi są przechowywane jako liczby całkowite długie – *long integer* (32 bity). Mnożenie wszystkich wag przez K jest równoważne mnożeniu z kakrokiem stałej C przez odwrotność K. Wówczas zmienia się waga tylko jednego znaku i jest już bardzo blisko omawianej wcześniej **własności rodzeństwa**. Trzeba jednak zastosować tu jej zmodyfikowaną wersję, gdyż w oryginale stosuje się ona tylko do wag zmieniających się zawsze o stałą wartość. Modyfikacja ta jest trochę złożona i nie będzie tu omawiana. Ponadto, co pewien czas następuje przekroczenie zakresu liczb *long integer* i trzeba przebudować całe drzewo od nowa.

Na koniec kilka słów o pewnych własnościach tego algorytmu. Niewidoczny liść ma tendencję do adaptowania się do aktualnego stopnia zmienności danych. Jeżeli w strumieniu źródłowym nie ma powtarzających się znaków, to niewidoczny liść otrzymuje bardzo dużą wagę i „okupuje" połowę drzewa. Wówczas kod każdego znaku jest wydłużany tylko o 1-bitowy kod Huffmana niewidocznego liścia. Z drugiej strony, jeżeli znaki się powtarzają i każdy kolejny pojawiający się znak rezyduje już w drzewie, to niewidoczny liść wędruje daleko „w dół" drzewa i praktycznie nie zajmuje miejsca.

Krzywa Hilberta

Warto także powiedzieć, w jaki sposób zredukować problem kompresji danych wielowymiarowych (np. dwuwymiarowego obrazu) do problemu jednowymiarowego. Lempel i Ziv [5] podali rozwiązanie optymalne w sensie informacyjno-teoretycznym.



Rys. 4. Krzywa Hilberta

Naiwne byłoby „skanowanie" obrazu wzdłuż kolumn lub wierszy; traci się bowiem wówczas informację o prostopadłym kierunku. Obrazy mają tendencję do bycia raczej plamami niż liniami. Rozwiązaniem jest pobieranie kolejnych piksli wzdłuż tzw. krzywej wypełniającej przestrzeń, jak np. krzywa Hilberta, której początek, w przypadku dwuwymiarowym przedstawiono na rysunku 4. Główną jej zaletą jest właściwość, że przestrzeń jest rekurencyjnie dzielona na kwadraty, a krzywa Hilberta po wejściu do każdego kwadratu nie wyjdzie z niego dopóki nie

natrafi na wszystkie jego piksle. Zachodzi to na każdym poziomie rekurencji.

W przypadku obrazów, których długości boków nie są potęgami 2, problem rozwiązuje się w ten sposób, że bierze się odpowiednio dużą krzywą Hilberta, pokrywającą cały obraz, a rozważa się tylko punkty pokrywające się z obrazem.

• • •

Praktycznie istnieją tylko dwa podstawowe podejścia do problemu kompresji danych bez utraty informacji: metoda Huffmana oraz Lempel-Ziv-Welcha. Stanowią one podstawę bardziej zaawansowanych algorytmów, których omówienie niejednokrotnie wymagałoby oddzielnego opracowania. W zakresie szczegółów implementacyjnych nikt jeszcze nie powiedział ostatniego słowa, i na dobrą sprawę, nie wiadomo jakiego ostatecznego stopnia kompresji można się spodziewać. Jednak już dziś zyski są na tyle oczywiste, iż trudno bez kompresji danych wyobrazić sobie dalszy rozwój oprogramowania.

LITERATURA

[1] Barnsley M. F., Sloan A. D.: A Better Way to Compress Images. BYTE, Vol. 13, No. 1, 1988 pp. 215 223

[2] Faller N.: An Adaptive System for Data Compression. Conference Record of the Seventh IEEE Asilomar Conference on Circuits and Systems, 1973, pp. 593 597

[3] Hamming R. W.: Coding and Information Theory, Prentice Hall, Englewood Cliffs (NJ)

[4] Huffman D. A.: A Method for Construction of Minimum-Redundancy Codes. Proceedings of the IRE 40, 1952, pp. 1098 1101

[5] Lempel A., Ziv J.: Compression of Two-Dimensional Images. Combinatorial Algorithms on Words. Springer-Verlag. Berlin, 1985, pp. 141 154

[6] Miller V. S., Wegman M. N.: Variations on a Theme by Lemped and Ziv. Combinatorial Algorithms on Words, Springer-Verlag, Berlin, 1985, pp. 131 140

[7] Storer J. A.: Data Compression. Methods and Theory. Computer Science Press, 1988, pp. 69 72

[8] Vaughan-Nichols S.: Saving Space. BYTE, Vol. 15, No. 3, 1990, pp. 237 243

[9] Welch T. A.: A Technique for High-Performance Data Compression, IEEE Computer 17:6, 1984 pp. 8 19

[10] Ziv J., Lempel A.: Compression of Individual Sequences Via Variable-Rate Coding. IEEE Transations on Information Theory, 24:5, 1978 pp. 530 536.

Wszystkie osoby, które nie zdążyły zaprenumerować INFORMATYKI na cały 1991 rok mogą ją zaprenumerować na II, III i IV kwartał bieżącego roku do 28 lutego, a brakujące numery z pierwszego kwartału – kupić w redakcji.

abaQ

PRZEDSIĘBIORSTWO ZASTOSOWAŃ ELEKTRONIKI

ul. Obywatelska 128/152, 94–104 Łódź, tel. 86-49-30

oferuje:

- Płytki drukowane jedno- i dwustronne z metalizacją;
- Wykonywanie klisz i projektów płytek drukowanych;
- Programatory EPROM, PROM, 8748 do IBM PC/XT/AT;
- Emulatory EPROM;
- Asemblery i deasemblery skrośne procesorów 8-bitowych;
- Analizatory układów logicznych, 32-kanały, 20 i 40 MHz;
- Falowniki trójfazowe 2,2 kW, zasilanie 1- lub 3-fazowe, regulacja kierunku i prędkości do 200%, wejścia sterujące.

0/14/90

SWEDEX UNIVERSAL

SWEDEX UNIWERSAL

Ltd

oferuje:

MODEMY DUPLEKSOWE DO TRANSMISJI DANYCH:

- pracujące na łączach telefonicznych komutowanych i dzierżawionych, jednoparowych, zgodnie z normami V.21, V.22, V.22 bis CCITT:
  MODEM 1230 P (0–300 b/s asynchr., 600/1200 b/s asynchr./synchr.)
  MODEM 2412 P (0–300 b/s asynchr., 1200/2400 b/s asynchr./synchr.)
  MODEM 2412 H (automatyczny – standard Hayes, 0–300 b/s asynchr., 1200/2400 b/s asynchr./synchr.);
- pracujące na łączach telefonicznych dzierżawionych, dwuparowych: Konwerter KN-9600 (modem typu baseband, 2400 4800 9600 b/s synchr.);
- pracujące na łączach telefonicznych dzierżawionych, jednoparowych.

Wszystkie modemy posiadają świadectwo homologacji Instytutu Łączności

- Konwerter RS-232 na pętlę prądową 20 mA (do 19 200 b/s asynchr./synchr., odległość do 2 km).

DO KOMPUTERÓW PC XT/AT:

- interfejsy pomiarowe IEC 625 (IEEE 488):
  SU-GPIB (do 10 kB/s),
  SU-GPIBF (do 500 kB/s);
- programator SU-PROG pamięci EPROM 2716-27512 i mikrokomputerów jednoukładowych MCS 48, MCS 51;
- pakiet transmisji synchronicznej BSC: umożliwia pracę IBM PC jako terminala ICL 7181, ICL 7020, IBM 2780, IBM 3780, IBM 3270;
- oprogramowanie emulacyjne stacji IBM 3780.

DO MERY 9150:

- Koncentrator Stanowisk Operatorskich KSO-4;
- Adapter Wyniesionego Stanowiska Operatorskiego ASO z modemami;
- Adapter Telegraficzny ATG-2.

DO EMC ODRA 1305:

- Adapter Telekomunikacyjny AT-8/MPX (zastępuje UPD)

TESTERY:

- KSO (systemy mikroprocesorowe na INTEL 8080 i 8085);
- V.24 (zgodny z CCITT V.24/V.28 i EIA RS-232 C).

OFERTY – ZAMÓWIENIA – INFORMACJE

SWEDEX UNIVERSAL Ltd
UL. WSPÓLNA 1, 05–75 WESOŁA
k. WARSZAWY
TELEKS 813935 swedex pl,
WARSZAWA TEL. (022) 15-33-65

ZO/238/90

SWEDEX UNIVERSAL

supra

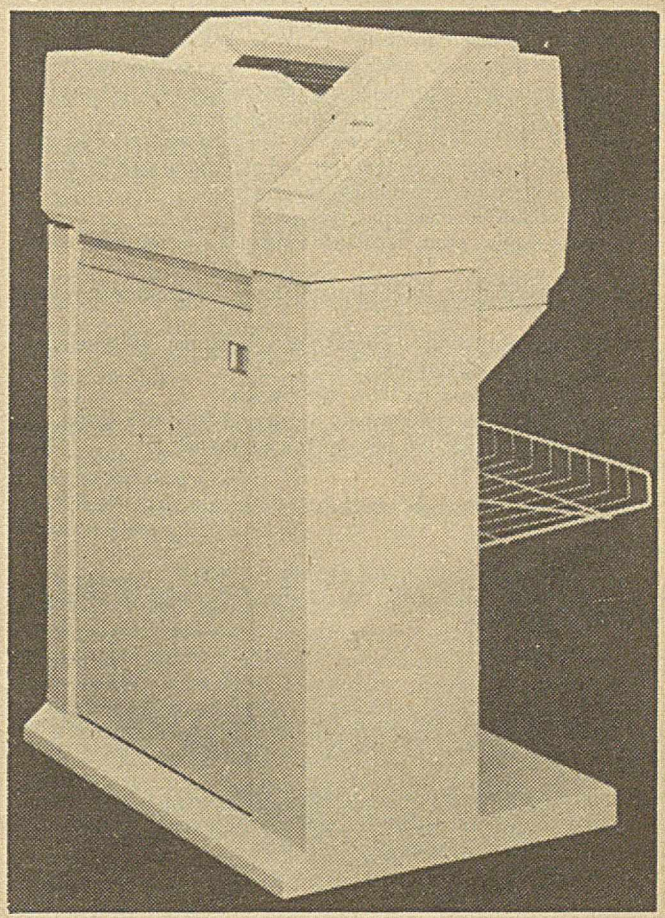
SUPRA SP. Z O.O.
ul. Trybunalska 54, 60-325 POZNAŃ
telefon: 674392, telefaks: 236498
teleks: 412966

OFERUJE ŚWIATOWY BESTSELLER

MANNESMANN
TALLY

DRUKARKĘ MOZAIKOWO-WIERSZOWĄ MT 645

- doskonała jakość druku
- niezawodność 7000 h MTBF
- duża szybkość 450 wierszy/min
- cicha praca
- bufor 27 KB
- liczba kopii 1+5
- pełna grafika 240 (dpi)
- polskie litery (opcja)
- wewnętrzny program testujący
- dwa interfejsy automatycznie przełączane
- podwójny system traktorów



MANNESMANN
TALLY

NAJPRAWDOPODOBNIEJ NAJLEPSZE
DRUKARKI NA ŚWIECIE

AUTORYZOWANY DYSTRYBUTOR DRUKAREK
MANNESMAN TALLY

supra

0/12/90

PIOTR FUGLEWICZ
Katowice

Systemy CASE przykłady narzędzi (2)

Liczba narzędzi, zaliczanych do kategorii CASE wynosi już w świecie co najmniej kilkaset i szybko rośnie. Dobry przegląd tych narzędzi zawierają dostępne w Polsce publikacje [12, 26]. Poniżej omówiono sześć programów, będących pierwszym krokiem w stronę polskiego CASE. Dwa z nich są oparte na języku zgodnym z dBase III, dwa kolejne na własnych bibliotekach metod dostępu, jeden jest narzędziem do specyfikowania systemów czasu rzeczywistego, wreszcie ostatni – programem makietującym z możliwością generowania kodu.

TurboGen

Generator aplikacji TurboGen zaprogramowano korzystając z systemu FoxBase w zespole programistów PKiWSI „WEKTOR" w Warszawie. Projektanci generatora mieli na uwadze dwa podstawowe cele:
- stworzenie narzędzia umożliwiającego generowanie w krótkim czasie dowolnego (w określonej klasie) systemu użytkowego,
- zapewnienie możliwie dużej pewności działania systemów użytkowych przez budowę tych systemów przy wykorzystaniu stałych, dobrze przetestowanych modułów biblioteki generatora.

Generator składa się z następujących modułów:
- modułu tworzenia i modyfikowania struktury systemu użytkowego w zakresie:
  - specyfikacji baz danych, ich struktur, budowy i wzajemnego powiązania,
  - określenia sposobu kontroli poprawności formalnej i merytorycznej pól przy wprowadzaniu danych,
  - tworzenia i inicjowania stałych systemowych,
- modułu budowy i modyfikowania menu systemu użytkowego,
- modułu obsługi fizycznych baz danych w zakresie:
  - dodawania rekordów,
  - przeglądania,
  - korekty,
  - usuwania,
- modułu obsługi logicznych baz danych w zakresie:
  - organizacji ekranu (położenie pól rekordu logicznego na ekranie),
  - dodawania rekordów,
  - przeglądania,
  - korekty,
  - usuwania.

(logiczna baza danych jest jedną bazą danych z punktu widzenia użytkownika, chociaż może składać się z kilku fizycznych baz powiązanych logicznie;
- modułu obsługi informacyjnej systemu użytkowego („help");
- modułu generowania różnego rodzaju raportów systemu, z możliwością agregacji danych na różnych poziomach;
- modułu zarządzania systemu w zakresie:
  - kopiowania baz danych,
  - bieżącej kontroli nad bazami, które uległy zmianie w czasie sesji systemu,
  - zabezpieczenia wykonywania poszczególnych operacji z menu systemu poprzez rozbudowany system haseł.

W generatorze, oprócz typów pól charakterystycznych dla standardu dBase, wyróżniono jako dodatkowy typ pole atrybutowe. Pole to różni się od pozostałych sposobem wprowadzania danych; jest to pole, które określa atrybut, mogący przybierać wartości ze skończonego zbioru, przy czym każdej wartości jest przypisana określona wartość atrybutu. Wprowadzanie danych odbywa się przez wybór zorganizowany w postaci menu.

Generator dzieli bazy danych na dwie rozłączne klasy. Do pierwszej należą tzw. bazy pojedyncze, do drugiej tzw. bazy logiczne. Baza pojedyncza jest bazą, która fizycznie reprezentuje jeden plik. Baza logiczna składa się z co najmniej dwóch baz pojedynczych związanych ze sobą logicznie przez określone wspólne pola. Baza logiczna umożliwia budowę hierarchicznych struktur baz danych. Menu główne generatora przedstawiono na rys. 1.

1. Specyfikacja bez danych systemu
2. Definiowanie wszystkich pól wszystkich baz systemu
3. Definiowanie treści menu pól atrybutowych
4. Określenie struktur baz danych
5. Zakładanie fizycznych plików baz danych
6. Specyfikacja baz logicznych
7. Automatyczna budowa programów obsługi baz danych
8. Budowa ekranów dla baz logicznych
9. Budowa menu systemu
10. Ustalenie położenia poszczególnych tablic (okienek) menu
11. Tworzenie dodatkowych plików indeksowych
12. Budowa raportów

Rys. 1

NanoX

Program Tworzenia Dialogowych Systemów Przetwarzania Danych NanoX zaprogramowano korzystając z systemu Clipper w zespole programistów PTH BISTER w Katowicach. NanoX jest narzędziem projektanta i programisty, tworzących dialogowy system przetwarzania danych w języku Clipper '87. Wspomaganie pracy przez ten program obejmuje:
- definiowanie dialogu tworzonej aplikacji; przyjęto, że aplikacja będzie miała dialog realizowany metodą hierarchicznych menu; w trakcie generowania dialogu kształt i efekty tego dialogu są symulowane;
- definiowanie nazw funkcji systemu (opisywanych w menu) oraz ich atrybutów, tj.: typu, poziomu ochrony oraz nazw procedur kontrolnych;
- określanie mechanizmu i treści podpowiedzi systemu aplikacyjnego;
- projektowanie w sposób dialogowy, z natychmiastową symulacją ekranów-formatek służących do wyświetlania informacji i pobierania danych od użytkownika aplikacji oraz wydawnictw – raportów prezentujących dane systemu;
- udostępnianie gotowych, standardowych funkcji systemów przetwarzania danych (redagowania danych, prezentacji danych, archiwowania danych);
- udostępnianie typowych elementów oprogramowania zebranych w bibliotekę procedur, zwłaszcza mechanizmów: słownikowego wspomagania redagowania pól kodowanych oraz kalkulatora wspomagającego redagowanie pól numerycznych;
- uwzględnianie w indeksowaniu właściwości dowolnie zdefiniowanego alfabetu;
- automatyczne tworzenie dokumentacji technicznej i użytkowej systemu;
- nadzór nad scalaniem systemu.

Efektem użycia programu NanoX jest zapis danych o projektowanym systemie aplikacyjnym w postaci Opisu Aplikacji. Pozwala to na:
- symulację, pod kontrolą programu NanoX, pracy systemu aplikacyjnego w trakcie jego projektowania;
- tworzenie makiety systemu aplikacyjnego; makieta realizuje wszystkie funkcje już zdefiniowane przez projektanta oraz symuluje wykonanie funkcji, których definicji jeszcze nie ukończono;
- generowanie ostatecznej postaci systemu aplikacyjnego;

– automatyczne tworzenie dokumentacji technicznej systemu aplikacyjnego.

Proces tworzenia aplikacji nie musi kończyć się w chwili jej wygenerowania. W dowolnej chwili (o ile zostanie przechowany Opis Aplikacji) można powtórzyć wszystkie fazy projektu, dopasowując aplikację do zmieniających się warunków jej stosowania. Dodatkową zaletą takiego rozwiązania jest możliwość użycia fragmentów gotowych systemów aplikacyjnych do generowania kolejnych aplikacji (o ile te systemy zostały stworzone za pomocą programu NanoX).

NanoX został pomyślany jako model systemu wspomagającego tworzenie aplikacji. Przyjęto, że zakres tworzonych przez ten model aplikacji ogranicza się do dialogowych systemów gromadzenia i przetwarzania danych. Dość proste struktury danych, których obsługę umożliwia Clipper, występują na tyle często, że zakres stosowania narzędzia jest dość szeroki.

Z funkcji systemów użytkowych wyodrębniono te, które występują najczęściej w podobnej postaci:
- dialog z użytkownikiem,
- redagowanie i prezentację danych zapisanych w systemie,
- przetwarzanie danych,
- ochronę funkcji (i danych) przed niepowołanym dostępem i utratą.

Przetwarzanie uznano za najbardziej indywidualną część każdego z systemów i przyjęto, że nie będzie ono wchodziło w zakres wspomagania tym narzędziem.

mixPACK

Biblioteka Relacyjnej Bazy Danych mixBASE i Generator Aplikacji mixUTIL są produktami PTH „KaNet" z Katowic. Biblioteka mixBASE zawiera zestaw procedur realizujących funkcje relacyjnej bazy danych. Właściwości bazy mixBASE można ująć najogólniej w następujące grupy:

- mixBASE umożliwia tworzenie baz relacyjnych. Wszystkie dane są przechowywane w relacjach pamiętanych w postaci indeksowo-sekwencyjnych plików dyskowych. Jednocześnie właściwości mixBASE w tym zakresie wykraczają poza możliwości tradycyjnych baz relacyjnych, dzięki udostępnieniu programiście struktury danych (nazywanej „plikiem wirtualnym"), łączącej informacje zawarte w wielu różnych relacjach, oraz dopuszczeniu wszystkich operacji (przeglądanie, dodawanie, modyfikacja i kasowanie) na takiej konstrukcji logicznej.

- Słownik Opisu Aplikacji, obsługiwany przez wewnętrzne mechanizmy oprogramowania bazy, składa się z:
– słownika pól systemowych, zawierającego opisy pól występujących w relacjach (plikach dyskowych) oraz pól pamięciowych,
– słownika relacji (plików dyskowych), opisującego zawartość i sposoby udostępniania informacji przechowywanych w plikach fizycznych dla potrzeb aplikacji,
– słownika plików wirtualnych, definiującego zawartość i sposoby udostępniania agregatów danych, podlegających faktycznemu przetwarzaniu w aplikacji,
– słownika formatek ekranowych, zawierającego opisy sposobów wizualizacji konstrukcji zdefiniowanych w słowniku plików wirtualnych,
– słownika menu sterowania aplikacją, opisującego wygląd i działania wszystkich menu aplikacji,
– słownika formatów raportów, opisującego sposoby drukowania informacji udostępnianych przez konstrukcje zdefiniowane w słowniku plików wirtualnych,
– słownika opisu struktury aplikacji, zawierającego informacje na temat powiązań między funkcjami danej aplikacji, plikami wirtualnymi i menu sterowania,
– słownika operatorów (użytkowników) aplikacji, opisującego system autoryzacji dostępu do poszczególnych funkcji danej aplikacji.

Struktury danych zdefiniowane w słownikach są w dużej mierze niezależne od programu realizującego aplikację, tzn. modyfikacja np. jednej z formatek ekranowych nie powoduje konieczności rekompilacji programu.

Aplikacje zbudowane z użyciem mixBASE wykorzystują następujące cechy biblioteki:

- mixBASE zawiera rozbudowany system podpowiedzi, zarówno tekstowych, jak i konstruowanych na podstawie informacji zawartych w bazie danych,
- każda aplikacja zrealizowana w standardzie mixBASE ma wbudowane elementy języka zapytań o właściwościach zbliżonych do QUERY BY EXAMPLE,
- komunikacja dowolnej aplikacji wykonanej w standardzie mixBASE z użytkownikiem jest zawsze realizowana przy użyciu stałego zestawu klawiszy funkcyjnych (tzn. określony klawisz w każdej aplikacji powoduje wykonanie zawsze takiej samej akcji) oraz z wykorzystaniem techniki rozwijanych menu i okien,
- mixBASE jest z założenia bazą przeznaczoną do pracy w systemach wielodostępnych oraz sieciowych (wbudowane mechanizmy blokady dostępu do pliku wirtualnego – a więc grupy plików fizycznych, oraz rozwiązywanie konfliktów przy próbie modyfikacji rekordu wirtualnego – a więc grupy powiązanych ze sobą rekordów fizycznych),
- mixBASE nie zawiera istotnych ograniczeń konstrukcyjnych (możliwość jednoczesnego otwarcia 127 plików fizycznych, maksymalna długość rekordu 64 KB), umożliwiając tworzenie bardzo dużych aplikacji,
- istnieje możliwość śledzenia użytkowania aplikacji w dzienniku (ang. *journal*), z wykorzystaniem informacji dotyczących przebiegu śledzenia do ewentualnego odtworzenia zawartości bazy na podstawie ostatniej kopii archiwalnej.

Podane w powyższym zestawieniu istotne właściwości funkcjonalne aplikacji wykonanych w standardzie mixBASE, takie jak:
– język zapytań,
– rozbudowany system podpowiedzi,
– standardowy sposób komunikowania się z użytkownikiem,
– rozwiązywanie konfliktów modyfikacji tych samych danych przez różnych użytkowników,
są realizowane wewnętrznie przez procedury biblioteki mixBASE na podstawie danych zawartych w słownikach i nie wymagają od programisty aplikacji żadnego nakładu pracy.

Oprócz omówionej biblioteki procedur relacyjnej bazy danych, w skład pakietu mixBASE wchodzą następujące standardowe programy wspomagające, dołączane do każdej aplikacji:
mixVIEW – umożliwiający przeglądanie zawartości plików fizycznych z uwzględnieniem ich struktury (m.in. są wyświetlane nazwy pól rekordu),
mixUSER – program pozwalający na dokonywanie przez użytkowników ograniczonych (przewidzianych przez projektanta aplikacji) zmian w zawartości słowników,
mixJOUR – program obsługi dziennika (dostarczany wraz z aplikacjami wykorzystującymi dziennik).

Korzystając z biblioteki mixBASE opracowano generator aplikacji mixUTIL, będący pakietem programów przeznaczonych do tworzenia i pielęgnowania aplikacji, realizowanych z wykorzystaniem biblioteki. Podstawowy zestaw programów pakietu mixUTIL jest przeznaczony do tworzenia i pielęgnowania słowników przechowujących opisy struktur danych tworzonej aplikacji. Definiowanie struktur danych odbywa się w maksymalnym zakresie metodą wykorzystywania podpowiedzi oraz bieżącej kontroli poprawności wprowadzanych danych. W całym zestawie programów obowiązuje jednolita forma komunikacji z użytkownikiem, w której wykorzystano technikę rozwijanych menu i okien. Do projektowania formatek ekranowych, menu sterowania przebiegiem aplikacji oraz formatów wydruków zastosowano technikę makietowania. Pozostałe części składowe pakietu mixUTIL można podzielić na następujące grupy:
- moduł generowania programów: składa się z generatora, który na podstawie informacji zawartych w słownikach tworzy szkielet programu aplikacji zapisany w specjalnej, uproszczonej notacji, oraz z preprocesora przetwarzającego tę notację na program w języku C,
- moduł dokumentatora aplikacji, tworzący szkielet dokumentacji konstrukcyjnej przez odpowiedni wydruk informacji zawartych w słownikach opisu struktury danych,
- moduł analizatora spójności słowników, pozwalający także na tworzenie tablic powiązań poszczególnych elementów struktur danych opisanych w słownikach,
- moduł integratora aplikacji, umożliwiający łączenie słowników opisujących struktury danych dwóch lub więcej aplikacji w jeden komplet lub też dekompozycję jednego kompletu na kilka mniejszych (co może

mieć zastosowanie przy niezależnym tworzeniu fragmentów jednej aplikacji przez różnych programistów).

SUPER

Generator systemów użytkowych SUPER powstał w warszawskiej firmie MacroSoft. Systemy użytkowe tworzone z użyciem generatora SUPER charakteryzują następujące cechy:

Relacyjność. Występujące bazy danych są strukturami tabelarycznymi, przetwarzanymi przez zamknięty język programowania. Język ten jest rozumiany przez jądro wykonawcze (ang. *engine*) tak, że polecenia tego języka na zbiorze wierszy bazy relacyjnej są wykonywane bez tłumaczenia na język proceduralny.

Utrzymywanie integralności danych. Zapewniono kontrolę dzięki tabelom indeksów, przechowywanym w plikach .NDX. Jest także utrzymywana niepowtarzalność kluczy w wybranych tabelach indeksów.

Obsługa przekrojów międzybazowych. Jest możliwa prezentacja i redagowanie danych z więcej niż jednej bazy danych, tworzących identyfikowany przez nazwę przekrój międzybazowy. Podczas redagowania jest kontrolowana integralność referencyjna, przy odpowiedniej definicji relacji.

Komunikowalność. Zapewniono możliwość importu i eksportu danych. Każdy system użytkowy jest wyposażany w mechanizm kontekstowego samouczka z możliwością jego redagowania przez użytkownika. Do aktualizowania danych służą zautomatyzowane funkcje przeglądania i redagowania, wspomagane mechanizmami obsługi menu. Jest możliwe działanie interakcyjne z użyciem oryginalnego języka, jak również tworzenie wydruków w języku generatora sprawozdań.

Korzystanie z katalogu systemu. Rolę katalogu systemu pełni tu tzw. plik sterujący .DEF. Zawiera on opisy wszelkich struktur baz danych, indeksów, relacji i komunikacji z użytkownikiem.

Odtwarzanie utraconych danych. Jest realizowane dla tabel indeksów (pliki .NDX). System ma funkcję odtwarzania indeksów uszkodzonych w wyniku nieprawidłowego przebiegu pracy lub na życzenie użytkownika.

Brak kontroli transakcji. Nie istnieje możliwość opisania i automatycznej kontroli transakcji.

Systemy użytkowe, tworzone przy użyciu generatora działają w ten sposób, że program wykonawczy .EXE działa na relacyjnych bazach danych (pliki .DBF), czerpiąc informacje opisujące dany system użytkowy z katalogu systemu (plik sterujący .DEF). Plik ten jest wynikiem generowania systemu, na podstawie projektu. Do takiego generowania służy SUPER, będący ponadto edytorem katalogu systemu. Do tworzenia algorytmów obliczeniowych dla systemów użytkowych stosuje się język Formula, którego wyrażenia są interpretowane podczas pracy systemu przez wbudowany interpreter. Kontaktowanie się poziomu algorytmu z rzeczywistym światem bazy danych zrealizowano przez zespół funkcji dostępu MacroBase. Umożliwiono także sporządzanie sprawozdań, które wykonują się według zapisu sterującego zawartego w plikach tekstowych, czytanych i interpretowanych przez język Report.

Program wykonawczy systemu użytkowego zawiera:
- procedury inicjowania pracy systemu użytkowego,
- procedury obsługi błędów czasu wykonania (urządzeń zewnętrznych, pamięci, przepełnienia arytmetycznego itp.),
- procedury obsługi urządzeń zewnętrznych (klawiatury, ekranu, drukarki itp.),
- funkcje dostępu do baz danych,
- jądro wykonawcze (engine, system przetwarzania relacyjnej bazy danych).
- interpreter języka formuł Formula,
- interpreter języka sprawozdań Report,
- dodatkowe procedury właściwe dla systemu użytkowego, wykonywane poza interpreterem formuł i funkcjami dostępu.

Użytkownik dostaje do dyspozycji uniwersalny program wykonawczy o nazwie MASTER.EXE, który czyta plik sterujący .DEF z dysku i wykonuje polecenia użytkownika korzystając z wbudowanych interpreterów. Aby system użytkowy rozpoczął pracę, wystarczy w wierszu polecenia systemu operacyjnego podać jako argument programu MASTER nazwę pliku sterującego .DEF. Po wstępnej fazie projektowej, pierwszym krokiem musi więc być redagowanie takiego pliku i jego zapisanie na dysk, tj. wygenerowanie systemu.

W systemach użytkowych MacroSoft wyróżnia się warstwy przedstawione na rys. 2.

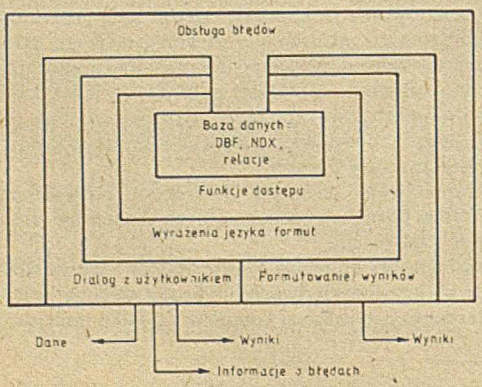
Informacje przepływają od świata zewnętrznego do bazy danych przez poszczególne warstwy, wyspecjalizowane w funkcjach podanych na rysunku. Warstwy te są realizowane przez odpowiednie moduły programowe zestawione w poniższej tabeli.

W ostatniej kolumnie tabeli podano czynności niezbędne do utworzenia systemu użytkowego dokonywane przy użyciu generatora SUPER.

Warstwa	Moduł	SUPER
Baza danych	Jądro wykonawcze	Definiowanie struktur baz danych, ich indeksów i relacji
Funkcje dostępu	MacroBase	
Wyrażenia języka formuł	MacroFormula	Definiowanie formuł sterujących systemu
Dialog z użytkownikiem	MacroWindows	Definiowanie okienek i formuł dialogu
Formatowanie wyników	MacroReport	
Obsługa błędów	MacroError	

Przetwarzanie informacji odbywa się dzięki wykonywaniu formuł języka Formula, przy czym mogą one być wywoływane jako skutek:

- Wyboru funkcji z menu. Formuła jest związana z pozycją w danym menu (formuła sterująca).
- Wyboru funkcji z wiersza podpowiedzi (formuła dialogu). Formuła, związana z klawiszami obsługi okienka wertowania.
- Zmiany położenia w bazie danych. Formuły wykonywane w czterech wypadkach (formuły dialogu):
przed wyświetleniem okienka (*Display*),
przed redagowaniem okienka (*Before edit*),
po redagowaniu okienka (*After edit*),
w wyniku dołączenia rekordu (*Blank*).
- Redagowania pola. Formuły wykonywane w trzech wypadkach (formuły dialogu):
przed wyświetleniem pola (*Display*),



Rys. 2

GŁÓWNE MENU
definicja Systemów
definicja Baz danych
definicja Relacji
definicja wolnych Okienek
definicja okienek Menu
definicja wolnych Pól
edycja Formuł
ratowanie Danych po awarii
tworzenie Zbiorów
scalanie Danych

Rys. 3

przed redagowaniem pola (*Before edit*),
po redagowaniu pola (*After edit*).

- Użycia w tekście wzorca sprawozdania .RPM, formuły wykonywane zgodnie z treścią wzorca.

W pierwszych trzech przypadkach formuły definiuje się z użyciem generatora SUPER. Główne menu generatora SUPER przedstawiono na rys. 3.

SPRITE

Opracowany w Instytucie Telekomunikacji Politechniki Warszawskiej program SPRITE jest środowiskiem wspomagającym stosowanie języka SDL. W zasadzie język SDL powstał dla potrzeb telefonii, ale jak już wcześniej wspomniano, zaczyna być coraz powszechniej stosowany przy specyfikacji różnych systemów czasu rzeczywistego. Cechami odróżniającymi SDL od innych języków są:
- istnienie dwóch równoważnych semantycznie wersji składni – graficznej SDL/GR i tekstowej SDL/PR,
- możliwość umieszczenia w specyfikacji elementów nieformalnych, co pozwala na stopniowe uszczegółowienie specyfikacji stosownie do zaawansowania projektu.

SPRITE jest jedynym z prezentowanych polskich narzędzi spełniających oba postulaty współczesnych programów klasy CASE, to znaczy zastosowanie sformalizowanej metody projektowania oraz graficzny sposób komunikacji z użytkownikiem. Podstawowym pojęciem systemu SPRITE jest projekt, obejmujący specyfikację wielu systemów z uwzględnieniem wielu ich wersji (odmian lub faz zaawansowania projektu). Każdemu obiektowi strukturalnemu w SDL (sieć, system, blok, proces) odpowiada oddzielny dokument składający się z arkuszy. Arkusz jest obszarem o wielkości określonej przez format A4, przeznaczonym do umieszczania diagramów.

Funkcje systemu SPRITE są realizowane przez zbiór powiązanych ze sobą modułów oprogramowania, obejmujący:
Konfigurator,
Edytor graficzny,
Edytor Tekstowy,
Analizator Składniowy,
Translator SDL/GR – SDL/PR,
Moduł Wydruku Dokumentacji,
Moduł Domknięcia Komunikacyjnego.

Interakcja graficzna z systemem SPRITE odbywa się przy użyciu myszy i jest bardzo wygodna dla użytkownika.

PROGEN

PROGEN, który powstał w gdyńskiej firmie SAMBA, obejmuje najskromniejszy zakres działań, jest bowiem generatorem kodu, opisującego strukturę menu aplikacji w powiązaniu z hierarchicznym systemem podpowiedzi. Warto jednak zwrócić nań uwagę ze względu na dwie cechy: najlepiej rozwiązany problem makietowania aplikacji (w trakcie jej tworzenia) oraz bardzo wygodny sposób komunikacji z użytkownikiem.

PROGEN generuje szkielet aplikacji w języku Turbo Pascal. Z generatorem są dostarczane jednostki (ang. *unit*), realizujące wiele, niekiedy bardzo wyrafinowanych, funkcji.

W chwili obecnej PROGEN nie ma podstawowego elementu dla systemów CASE – słownika aplikacji. Według informacji uzyskanych w firmie SAMBA trwają prace w kierunku wprowadzenia tego elementu. Z przedstawionych produktów PROGEN ma najelegantszy (obok systemu SPRITE, który pracuje w grafice) sposób komunikacji z użytkownikiem.

• • •

Przedstawione przykłady są dowodem na to, że również w Polsce pojawiła się potrzeba stosowania narzędzi klasy CASE. Warto zwrócić uwagę, że tylko jedno z przedstawionych narzędzi powstało w środowisku akademickim. Pozostałe opracowano w firmach, które mają nadzieję uzyskać większą wydajność pracy nad konkretnymi aplikacjami. Oczywiście jest to dopiero pierwszy krok w stronę CASE, ale dowodzący głęboko praktycznych korzeni systemów tej klasy.

Posiadasz maszyny
typu IBM, serii SM, PDP-11, MERA400...,
pracujące pod systemami operacyjnymi:

UNIX, XENIX, RSX, PC-MOS, MULTILINK, CONCURRENT PCDOS, QNX, CROOK...

To z całą pewnością potrzebujesz

TERMINALE ALFANUMERYCZNE

MT220

Skontaktuj się z nami.

MICRONET

Zakłady Elektroniczne
"MICRONET"
81-836 Sopot ul. Krasickiego 9
tel. 51-13-17 tlx 051-2876 mcrnt

0/17/98

ZAPRASZAMY

TERA Spółka Z O. O.

Przedsiębiorstwo Popierania Postępu TERA Spółka z o.o. uprzejmie informuje, że posiadając kilkuletnie doświadczenie w instalacji systemów wspomagających zarządzanie

o f e r u j e :

- opracowanie projektów systemów,
- optymalny kosztowo i rozwojowo dobór sprzętu i oprogramowania,
- instalację systemu u klienta,
- bezpłatnie przez rok konserwację oprogramowania oraz serwis sprzętu wraz z doradztwem techniczno-eksploatacyjnym,
- szybką dostawę uzupełnień konfiguracji lub sprzętu mikrokomputerowego niezależnego od systemu, w tym mikrokomputerów ALR firmy Wearnes Technology.

Wszelkie dodatkowe informacje uzyskają Państwo codziennie oprócz niedziel w Biurze Handlowym
40-025 Katowice, ul. 27 Stycznia 31/17
tel. (faks): 51-61-65, teleks: 315448 tera pl

PAMIĘTAJ! Instalacje XENIX/NOVELL/PC MOS 386 oraz serwis to nasza specjalność – ZAPRASZAMY, ponieważ czterech lat doświadczeń nigdzie nie kupisz.

0/19/90

O standardzie ANSI języka C (1)

Za pierwszy standard języka C należy uznać książkę Kernighana i Ritchiego „The C Programming Language" wydaną w 1978 roku. Język C, pomyślany jako język programowania systemowego, zaczął od tej chwili robić niesłychaną karierę. We wczesnych latach osiemdziesiątych zaczęły pojawiać się na rynku pierwsze kompilatory języka C przeznaczone do użytku posiadaczy komputerów osobistych. W połowie dekady kompilatory C różnych firm stały się dostępne dla prawie każdego użytkownika niezależnie od tego na jakim sprzęcie pracował i jaki system operacyjny miał zainstalowany.

Naturalnym następstwem tej sytuacji stała się kwestia przenoszenia oprogramowania napisanego w języku C pomiędzy różnymi implementacjami. I tu wyszło szydło z worka: okazało się, że ten sam program w C przetłumaczony za pomocą różnych kompilatorów zachowuje się zupełnie inaczej.

Głównym powodem takiego stanu rzeczy był fakt, że książka Kernighana i Ritchiego (uważana, i słusznie, za jeden z najlepszych podręczników języka C) zawiera sporo niejednoznaczności różnie interpretowanych przez autorów kompilkatorów. Innym powodem, równie istotnym, było wprowadzenie do niektórych implementacji rozszerzeń języka C oraz „udogodnień" zorientowanych na konkretne rozwiązania sprzętowe.

W 1984 roku powstał w USA Komitet Standaryzacji Języka C mający za zadanie opracowanie normy ANSI języka C. Oprócz formalnego opisu składni i semantyki języka C, Komitet miał również na celu standaryzację funkcji bibliotecznych języka C oraz typów i makrodefinicji zawartych w plikach nagłówkowych. Standard zawiera też wymagania, które spełniać musi kompilator języka C. Materiałem wyjściowym do opracowania standardu ANSI był podręcznik Kernighana i Ritchiego. Zakładając, że Czytelnikowi znany jest wspomniany podręcznik, przedstawimy tutaj najbardziej istotne zmiany wprowadzone przez normę ANSI.

Umiędzynarodowienie Języka C

Alfabetem języka C jest zbiór znaków ASCII (precyzyjniej, reprezentacja graficzna znaków ASCII). Stanowi to problem poza granicami USA, w krajach gdzie niektóre znaki standardu ASCII zamienione zostały przez znaki narodowe. Wychodząc naprzeciwko temu problemowi, podjęto decyzję, że określone znaki mogą być zastąpione przez trzyznakowe sekwencje – trigrafy (ang. trigraph).

trigraf	??=	??(	??/	??)	??'	??<	??!	??>	??-
znak	#	[	\	]	^	{	|	}	~

Preprocesor

Norma ANSI zakłada, że preprocesor zna pięć makrodefinicji:

DATE – data kompilacji,
FILE – nazwa kompilowanego pliku,
LINE – numer aktualnego wiersza w kompilowanym pliku,
STDC – określa, czy implementacja spełnia normę ANSI,
TIME – godzina, minuta i sekunda kompilacji.

Dodano nowe dyrektywy preprocesora: # **elif**, # **error**, # **pragma**, i # (dyrektywa pusta). Pierwsza z nich umożliwia stosowanie mechanizmu „else-if" przy kompilacji warunkowej, druga powoduje wypisanie przez kompilator komunikatu, trzecia zleca kompilatorowi wykonanie pewnych akcji (zbiór możliwych akcji zależy od konkretnej implementacji). Jednocześnie wprowadzono zasadę, że w dyrektywie preprocesora przed i po znaku # może wystąpić ciąg „białych znaków" (odstępy, tabulacje itd.).

W wyrażeniach stałych, występujących w dyrektywie **if**, może występować jednoargumentowy operator **defined** oraz operatory logiczne: jednoargumentowy! i dwuargumentowe && i ||.

Przykład 1

Kompilacja pliku **predef.c** zawierającego program

```
main()
{
#error Kompilacja pliku
#if defined __STDC__ && __STDC__
    printf("ANSI C. %s: %d: %s, %s\n",__FILE__,__LINE__,
        __DATE__,__TIME__);
#else
    printf("Nie ANSI C.\n");
#endif
}
```

nastąpiła 15 listopada 1990 roku o godz. 11:30. W czasie kompilacji (o ile kompilator akceptuje normę ANSI) zostanie wypisany tekst

Kompilacja pliku

Wykonanie programu w dowolnym czasie spowoduje wypisanie tekstu

ANSI C. predef.c: 5: Nov 15 1990, 11:30:24

Przykład 2

Zamiast zagnieżdżonych, mało czytelnych dyrektyw warunkowych

```
#ifdef I80386
#define REG 32
#else
#ifdef I80286
#define REG 16
#else
#ifdef I8086
#define REG 16
#else
#define REG 8
#endif
#endif
#endif
```

standard ANSI C pozwala na napisanie

```
#ifdef I80386
    #define REG 32
#elif defined I80286 || defined I8086
    #define REG 16
#else
    #define REG 8
#endif
```

Przykład 3

W czasie tłumaczenia fragmentu programu

```
#pragma loop_opt(on)
    for(;;){ ...
    }
#pragma loop_opt(off)
    for(;;){ ...
    }
```

kompilator Microsoft C 5.1 (akceptujący w dużej mierze normę ANSI) spróbuje optymalizować kod pierwszej pętli, nie optymalizując kodu drugiej.

Najbardziej istotną zmianą w preprocesorze wydaje się wprowadzenie dwóch nowych operatorów # i # # do aparatu makrodefinicji. Operator # z argumentu makrodefinicji czyni stałą tekstową. Operator sklejenia # # z dwóch atomów leksykalnych tworzy jeden.

Przykład 4

Po wystąpieniu makrodefinicji

```
#define suma(x) printf("suma %s daje %d\n",#x,x)
#define zmienna(x) printf("%d\n",zm ## x)
```

tekst

```
suma('a'+5);
zmienna(A) ; zmienna(12);
```

zostanie zastąpiony przez preprocesor języka C tekstem

```
printf("suma %s daje %d\n","'a'+5",'a'+5);
printf("%d\n",zmA) ; printf("%d\n",zm12);
```

Leksyka

W celu uniknięcia niejednozaczności przyjęto zasadę, że atom leksykalny jest tworzony przez najdłuższy, możliwy ciąg znaków akceptowany przez analizator leksykalny. Tekst **a+++b** jest interpretowany jako ciąg atomów leksykalnych: **a, ++, + i b.** Ustalono, że komentarz jest równoznaczny jednemu znakowi odstępu. Dopuszczono jeden nowy operator – jednoargumentowy +.

Ze zbioru słów kluczowych usunięto **entry**, a dodano pięć nowych: **const, enum, signed, volatile** i **void.** Znaczenie tych słów zostanie wyjaśnione w dalszej części tekstu.

Zmodyfikowano dotychczasowy zapis przyrostka w stałych całkowitych przez możliwość dopisania **u** lub **U** oznaczającego, że stała jest typu **unsigned**. Przyrostek może być teraz jedną z liter **l, L, u, U** lub kombinacją **l** albo **L** z **u** albo **U**.

Przykład 5

Stała **123UL** oznacza długą stałą całkowitą bez znaku. Zakładając, że **n** zostało zadeklarowane jako **int**, wyrażenie

n + 123ul

jest typu **long unsigned**.

Stała zmiennoprzecinkowa może mieć dwa rodzaje przyrostków: **f** lub **F** co oznacza pojedynczą precyzję albo **l** lub **L** oznaczające poczwórną precyzję. W tym miejscu konieczne jest wyjaśnienie: „pojedyncza precyzja" oznacza precyzję nie większą niż „podwójna precyzja", a ta z kolei oznacza precyzję nie większą niż „poczwórna precyzja".

Rozszerzono notację stałych znakowych. Stała znakowa '\a' reprezentuje sygnał dźwiękowy, a stała '\v' znak tabulacji pionowej. Dopuszczono możliwość reprezentacji szesnastkowej stałych znakowych w postaci '\xh, '\xhh lub'\xhhh, gdzie h oznacza cyfrę szesnastkową (cyfry i małe lub duże litery od a do f). Powyższe rozszerzenia dotyczą też stałych tekstowych. Wystąpienie obok siebie stałych tekstowych (rozdzielonych, być może, „białymi" znakami) spowoduje potraktowanie ich jako jednej stałej tekstowej.

Przykład 6

Wyrażenie

t = "\x54ekst1" /*sklejenie*/ "Tekst2\a\n"

jest równoważne wyrażeniu

t = "Tekst1 Tekst2\007\n"

Typy

Niżej zamieszczamy pełny wykaz podstawowych typów danych; typy wymienione w jednym wierszu są synonimiczne.

```
char
signed char
unsigned char
short int, short, signed short int, signed short
unsigned short, unsigned short int
int, signed int
unsigned int, unsigned
long int, long, signed long, signed long int
unsigned long int, unsigned long
float
double
long double
```

Norma ANSI wprowadza nowy atrybut typów całkowitych – **signed** (ze znakiem) – mający, poza jednym wyjątkiem, znaczenie raczej dekoracyjne. Tym wyjątkiem było pozostawienie do uznania autorów kompilatorów decyzji, czy typ **char** reprezentuje wartości dodatnie i ujemne, czy tylko nieujemne. Najczęściej był to problem, czy zmienna typu **char** przyjmuje wartości ze zbioru [–128, 127], czy ze zbioru [0,255]. Norma ANSI pozostawia implementacji decyzję, czy typ **char** jest równoważny typowi **signed char** (wartości dodatnie i ujemne) czy **unsigned char** (wartości nieujemne). Pozostałe wymienione typy (ze znakiem lub bez) muszą spełniać warunki:

```
sizeof(short) <= sizeof(int) <= sizeof(long)
sizeof(float) <= sizeof(double) <= sizeof(long double)
```

Pojawiły się nowe atrybuty typów: **const** i **volatile**. Zmienna dowolnego typu z atrybutem **const** nie może mieć zmienianej w programie wartości (implementacja może, choć nie musi, umieszczać obiekt z atrybutem **const** w rejonie pamięci tylko do odczytu). Atrybut **volatile** oznacza, że obiekt może mieć zmienianą wartość poza kontrolą programisty. Może mieć to miejsce przy odwoływaniu się do tych rejonów pamięci, do których ma jednocześnie dostęp system operacyjny (asynchroniczne wejście/wyjście, przerwania). Jest to wskazówka dla kompilatora, by np. nie próbował – w celach optymalizacyjnych – utrzymywać obiektu tego typu w rejestrach.

Przykład 7

Zmienna **zegar_syst** zadeklarowana

extern long volatile const zegar_syst;

może być modyfikowana przez system operacyjny poza kontrolą programu. Jednocześnie zamienna **zegar_syst** jest dostępna tylko do odczytu.

Wyprowadzono typ wyliczeniowy. Składnia specyfikatora typu wyliczeniowego jest podobna do składni specyfikatora typu strukturowego.

Przykład 8

We fragmencie programu

```
enum kolor { zielony, niebieski, czerwony=10, czarny };
enum kolor barw = niebieski;
int n;
/* ... */
n = czarny + 1 - niebieski;
```

zdefiniowano typ wyliczeniowy **kolor** o czterech wartościach, równych odpowiednio liczbom całkowitym 0, 1, 10 i 11. Nazwy **zielony, niebieski, czerwony** i **czarny** mogą wystąpić w każdym kontekście, w którym może wystąpić stała całkowita.

(cdn)

WŁADYSŁAW MAJERSKI

Plan 9, czyli ucieczka do przodu

Prawdopodobnie mało kto wie o tym, że UNIX w AT&T to nie tylko UNIX System Laboratories i komercyjny **System V**. To również Bell Laboratories i słynny **UNIX Time-Sharing System** – historycznie rzecz biorąc pierwszy i przez długie lata główny wariant systemu. Wbrew powszechnemu mniemaniu wersje szósta (**V6**) i siódma (**V7**) bynajmniej nie zakończyły tej linii rozwoju UNIX-a, tyle tylko, że następne edycje[1] – z najnowszą **Tenth Edition** włącznie – powstawały już wyłącznie w celach badawczych (tzw. **Research UNIX**) i nigdy nie były rozpowszechniane.

Pierwsze pogłoski o nowym systemie operacyjnym pracowywanym w Bell Labs pojawiły się z górą rok temu, lecz początkowo nie wzbudzały większego zainteresowania – wydawało się, że chodzi po prostu o kolejny UNIX z tego źródła. A jednak wkrótce okazało się, że pogłoski były prawdziwe: w Bell Labs rzeczywiście powstaje zupełnie nowy system o tajemniczej nazwie **Plan 9**[2] [1, 2, 3], a wśród jego autorów są legendarni Dennis Ritchie i Ken Thompson!

Co mogło skłonić twórców UNIX-a do porzucenia – choćby tylko chwilowego – swego systemu? Odpowiedź na to pytanie kryje się w szczegółach koncepcji Planu 9. Warto poświęcić im trochę uwagi: być może tak właśnie wyglądałby dziś UNIX, gdyby zależało to tylko od jego autorów.

ARCHITEKTURA SYSTEMU

Plan 9 to wielodostępny system rozproszony, projektowany z myślą o konfiguracjach składających się z setek czy nawet tysięcy komputerów i terminali, który ma stać się w przyszłości głównym **środowiskiem do tworzenia oprogramowania** dla pracowników Bell Labs. Podstawowym elementem koncepcji systemu jest sprzętowe **rozdzielenie funkcji przetwarzania i przechowywania** danych [3]: każdą z tych usług świadczą inaczej skonfigurowane komputery.

Komputery przetwarzające (4 procesory, 128 MB pamięci, żadnych dysków) i **przechowujące** (1–2 procesory, 64 MB pamięci, dysk magnetyczny 600 MB i optyczny 300 GB) łączy się (korzystając bezpośrednio z DMA, 20 Mb/s) w pary lub w liczniejsze zestawy, a zestawy między sobą tradycyjnymi sieciami typu Ethernet lub Datakit. Użytkownicy korzystają ze specjalnie zaprojektowanych **inteligentnych terminali** (M68020, 4–8 MB pamięci, ekran 1024 × 1024/2 bity na piksel, klawiatura,mysz), dołączanych do sieci lokalnie (1–2 MB/s) lub zdalnie. Terminale nie przechowują danych, a z ich inteligencji korzysta głównie system – poza edycją tekstów w zasadzie całe przetwarzanie użytkowe odbywa się zdalnie. Pamięć operacyjna komputerów przechowujących pełni rolę podręcznej pamięci plików; wędrówka plików pomiędzy pamięcią i dyskami komputerów przechowujących nie angażuje komputerów przetwarzających, co więcej – jest dla nich całkowicie niewidoczna.

OPROGRAMOWANIE

Pod względem oprogramowania w Planie 9 niemal wszystko jest nowe: system operacyjny, oprogramowanie sieciowe, system okien, a nawet interpretator poleceń [4]. Po UNIX-ie nowy system dziedziczy wiele, lecz jest prostszy i nie zawiera większości z modnych ostatnio nowinek – na przykład nie ma lekkich procesów (wątków), procesy nie migrują i– o zgrozo! – nie korzysta się z *X Windows*.

W koncepcji systemu kluczowe znaczenie ma pojęcie **usługi**: świadczenie usług na rzecz procesów jest podstawową funkcją systemu. Wszelkie usługi, lokalne i zdalne, reprezentowane są w systemie plików jako **podkatalogi**, a korzystanie z usług sprowadza się do operowania na plikach. Na przykład, usługę drukowania tekstów można realizować jako katalog, w którym procesy tworzą pliki przeznaczone do druku, zaś usługę polegającą na udostępnieniu fragmentu systemu plików reprezentuje poddrzewo odpowiadające temu fragmentowi.

Każdy proces operuje w swojej przestrzeni nazw, którą tworzy suma nazw tych usług, z których proces może korzystać (procesy mogą z własnej inicjatywy dołączać usługi do swojej przestrzeni nazw). Innymi słowy, oznacza to, że każdy proces widzi własny, „wyimaginowany" system plików – sobie właściwy zlepek fragmentów rozmaitych „realnych" systemów plików – i korzysta tylko z tych usług, które są w nim reprezentowane. Tworzenie takich indywidualnych wyobrażeń procesów użytkownika o systemie plików jest jedną z ważniejszych funkcji systemu operacyjnego terminala.

Interesujący, nie spotykany dotąd w UNIX-ie rodzaj usługi reprezentują **podkatalogi procesów** (sam pomysł nie jest nowy, por. [5]). Dla każdego z lokalnych procesów system tworzy odrębny podkatalog, grupujący pliki dynamicznie odwzorowujące stan tego procesu: plik **mem** udostępnia obraz pamięci procesu, plik **status** – jego stan, plik **text** zawiera kod jego programu, zapis do pliku **ctl** umożliwia sterowanie procesem, a plik **note** umożliwia komunikowanie się z nim.

W Planie 9 usługi udostępniania plików mogą realizować również programy użytkowe. Przykładem tego jest **system okien**, którego rola sprowadza się do udostępniania procesom związanym z różnymi oknami prywatnych egzemplarzy pliku **/dev/cons** (między /dev/cos i UNIX-owym /dev/tty jest pewna analogia, tyle że plikiem /dev/cons nie zarządza system operacyjny). Procesy wiążą z tym plikiem swe standardowe wejście i wyjście, a system okien symuluje jego pożądane cechy. Ciekawą konsekwencją takiej realizacji jest to, że system okien może być swym własnym klientem.

Rejestrując się w systemie użytkownik najpierw instruuje terminal, z którym komputerem przechowującym pliki chciałby pracować. Terminal ściąga i uruchamia swój system operacyjny, który z kolei wykonuje polecenia zawarte w pliku *profile* użytkownika, w tym m.in. polecenia umieszczające w lokalnej przestrzeni nazw usługi, zwykle wyorzystywane przez tego użytkownika. Potem zazwyczaj startuje system okien i interpretator poleceń. Programy poleceń może wykonywać sam terminal lub dowolny, wybrany przez użytkownika komputer przetwarzający; w każdym przypadku w ramach tej samej lokalnej przestrzeni nazw.

Autorom systemu udało się uzyskać zdumiewająco małe rozmiary systemu: jądro ma wszystkiego 454 wiersze w asemblerze i 4421 wierszy w języku C (w tym 774 wiersze w plikach nagłówkowych); do tego dochodzi 1020 wierszy w C realizujących sprzęg z 29 funkcjami systemowymi. Usługi systemowe i oprogramowanie sieciowe to dalsze 9511 wierszy w C. I porównajmy to teraz z ponad milionem wierszy w ostatniej wersji Systemu V!

* * *

Plan 9 jest w pewnym sensie próbą powrotu do źródeł. Podobnie jak dwadzieścia lat temu w przypadku UNIX-a, jego projektantom i tym razem chodziło o stworzenie systemu dla siebie: prostego i wygodnego środowiska do tworzenia oprogramowania. Rozczarowani współczesnym UNIX-em, ich zdaniem zbyt ciężkim, nieeleganckim i skomplikowanym, za sprawą wszechogarniającej działalności standaryzacyjnej (por. *INFORMATYKA* 11–12 1990) praktycznie pozbawieni możliwości dalszego wpływania na jego rozwój, wybrali ruch być może najlepszy z możliwych – ucieczkę do przodu.

LITERATURA

[1] Duff T.: Rc – A Shell for Plan 9 and UNIX Systems. EUUG Newsletter. Vol. 10, No. 3, pp. 12–22, Autumn 1990

[2] Goos A.: Is Plan 9 Sci-Fi or UNIX for the Future? UNIXWORLD. Vol. 7, No. 10, pp. 61–62, October 1990

[3] Killian T. J.: Processes as Files. Proc. of the USENIX Conference, pp. 203–207, Summer 1984

[4] Pike R. i in.: Plan 9 from Bell Labs. EUUG Newsletter. Vol. 10, No. 3, pp. 2–11, Autumn 1990

[5] Ritchie D.: On Plan 9. BYTE, p. 312, September 1990.

JACEK SURMA

Przypominamy Państwu, czekamy na listy z uwagami i propozycjami tematycznymi do działu /unix.

UNIX jest zastrzeżonym znakiem towarowym AT&T.

[1] Tradycyjnie w ramach tego wariantu UNIX-a numeracja wersji (edycji) systemu wynika z numeracji wydań publikowanej dokumentacji.

[2] Anglosasi wiedzą, że nazwę wzięto z tytułu popularnego w latach pięćdziesiątych filmu science-fiction ***Plan 9 From Outer Space***.

HANNOVER MESSE CeBIT '91
World Center Office · Information · Telecommunications
13 – 20 MARCH 1991

CeBIT'91

Międzynarodowe targi technologii biurowej, informatycznej i telekomunikacyjnej CeBIT, wyodrębnione w 1986 r. z Hanowerskich Targów Przemysłowych w imprezę samodzielną, odbędą się w tym roku w dniach od 13 do 20 marca. CeBIT ma już ponad 20-letnią tradycję i na wiele lat przed swym usamodzielnieniem zdobył pozycję największej na świecie ekspozycji informatycznej, dystansując pod względem powierzchni, liczby wystawców oraz zwiedzających największe tego typu imprezy, takie jak amerykański COMDEX czy paryski SICOB.

Usamodzielnienie CeBIT nastąpiło w chwili, gdy mimo objęcia ponad połowy powierzchni wystawowej hal Targów Przemysłowych, z braku miejsca nie można było uwzględnić zgłoszeń setek nowych wystawców. Ten wywołany rewolucją mikrokomputerową rozwój był najbardziej chyba spektakularnym dowodem uzyskania przez technologię informatyczną zdecydowanej dominacji nad tradycyjnymi obszarami działalności przemysłowej.

Silnie tłumiony z braku powierzchni popyt wystawców wyrobów i usług tej technologii spowodował, że po usamodzielnieniu CeBIT nastąpił wprawdzie przeszło 50% wzrost powierzchni wystawienniczej, ale w dalszym ciągu nie można było zaspokoić wszystkich, coraz szybciej napływających nowych zgłoszeń (konieczna była adaptacja następnych hal wystawowych, dostosowanych dotąd do ekspozycji takich wyrobów, jak np. klasyczne obrabiarki). Trend niezaspokojonego popytu wystawców utrzymuje się po dzień dzisiejszy, mimo że w ciągu ostatnich dwu lat w wyniku intensywnego inwestowania przez Zarząd Targów, powierzchnia wystawiennicza CeBIT wzrastała o przeszło 20% rocznie. W roku bieżącym powierzchnia ta obejmie aż 395 000 m² brutto (286 100 m² netto), co w zestawieniu z początkami tej imprezy stanowi przeszło pięciokrotny wzrost. Punktem centralnym ekspozycji pozostaje w dalszym ciągu wybudowana w 1970 r. gigantyczna hala nr 1, która w pierwotnych założeniach miała na wiele lat zaspokoić potrzeby wystawiennicze przemysłu informatycznego. Obecnie, mimo że zachowała ona pozycję największej na świecie hali wystawowej, stanowi zaledwie ok. 20% łącznej powiechrzni CeBIT.

W jeszcze większym stopniu wzrasta liczba wystawców, która po usamodzielnieniu się imprezy natychmiast wzrosła o ok. 70%, a od tego czasu aż dwukrotnie, dochodząc w tym roku do niewiarygodnej wprost liczby 4553 firm, w tym 1772 firm zagranicznych z 45 krajów. W zbliżonym tempie wzrastała liczba zwiedzających, których w ubiegłym roku zarejestrowano ponad 561 tys., w tym ponad 113 tys. z zagranicy. Przy dotychczasowym trendzie wzrostu (a należy spodziewać się nawet znacznego przyspieszenia wskutek zjednoczenia Niemiec) nastąpi niewątpliwie przekroczenie w tym roku granicy 600 tysięcy zwiedzających (średnio 75 tys. dziennie). Taka liczba w połączeniu z kilkudziesięcioma tysiącami osób personelu wystawców zaczyna stwarzać istotne problemy dla infrastruktury Hanoweru, zwłaszcza w dziedzinie zakwaterowania.

Przy tak gigantycznych rozmiarach ekspozycji (17 dużych hal wystawowych, często wielokondygnacyjnych) istotne znaczenie dla zwiedzającego ma stale doskonalony od wielu lat podział tematyczny ekspozycji, niezbędny przy współczesnym podziale specjalizacji informatycznych. Doskonałym instrumentem, pomagającym orientować się na obszarze kilkudziesięciu hektarów ekspozycji i dokładnie lokalizować konkretnych wystawców, jest działający również od wielu lat, a dostępny dosłownie na każdym kroku, komputerowy system informacyjno-wyszukiwawczy.

Wspomniany podział tematyczny ekspozycji jednak nie został i prawdopodobnie nie będzie mógł być doprowadzony do końca, głównie ze względu na istniejący sposób zagospodarowania stoisk we wspomnianej już hali nr 1, w budowie której finansowo partycypowali przyszli wystawcy, uzyskując w ten sposób prawo do stałego miejsca ekspozycji. Odnosi się to zwłaszcza do dużych firm o zasięgu światowym, których większość całość swej, z reguły bardzo zróżnicowanej tematycznie, oferty demonstruje wyłącznie w ramach wspomnianych stałych stoisk, mimo że znaczna część tej oferty kwalifikuje się do tematyki innych hal. Charakter hali nr 1, zakwalifikowanej formalnie do tematyki „systemów informacyjnych i biurowych", ma jednak tę zaletę, że umożliwia zwiedzającemu wszechstronne zapoznanie się z kompletną ofertą światowej czołówki producentów sprzętu komputerowego. Pozostałym halom CeBIT udało się w zasadzie uzyskać charakter ekspozycji tematycznej, jakkolwiek granice podziału przebiegają w kilku przypadkach wewnątrz budynku (podział według kondygnacji).

Tegoroczne ekspozycje tematyczne CeBIT będą zlokalizowane następująco:

- komputeryzacja banków oraz instytucji finansowych: hale nr 2 i 18,
- bezpieczeństwo danych: hala nr 18,
- oprogramowanie, banki danych i doradztwo: hale nr 3 i 4,
- urządzenia peryferyjne: hale nr 5, 6 i 8,
- automatyzacja biur: hale nr 4 i 5,
- systemy dla małych przedsiębiorstw i mikrokomputery: hale nr 5, 6, 7 i 8,
- sieci komputerowe: hala nr 12,
- telekomunikacja: hale nr 13, 16, 17 i 23,
- prace badawcze i doradztwo innowacyjne: hala nr 23,
- techniki C (CAD, CAM, CIM, CAE, CAQ): hale nr 19 i 20,
- zarządzanie danymi produkcyjnymi: hala nr 18,
- szkolenie, doskonalenie, zarządzanie personelem, technika audiowizualna: hala nr 22.

Podczas CeBIT '91 w zlokalizowanym na terenie targowym Centrum Kongresowym odbędzie się również ponad 150 imprez towarzyszących (kongresy stowarzyszeń, seminaria, wykłady, odczyty), a także zostanie zorganizowanych sześć ekspozycji o charakterze promocyjno-propagandowym, nawiązujących do poszczególnych ekspozycji tematycznych (automatyzacja banków, bezpieczne przetwarzanie, europejskie sieci komputerowe, komunikacja satelitarna, perspektywy kariery zawodowej w 2000 r., nowoczesne formy przemysłowe).

Warto zauważyć, że Hanower znajduje się zaledwie ok. 300 km od naszych granic stwarzając dla nas szczególnie sprzyjające warunki do bezpośredniego zapoznania się z najbardziej wszechstronną ofertą światowych osiągnięć technologii informatycznej i telekomunikacyjnej.

WŁADYSŁAW KLEPACZ

/etc

Z UNIX Software Operation (USO) na **UNIX System Laboratories Inc.** (USL) zmienił nazwę dział AT&T zajmujący się produkcją, rozwojem, sprzedażą i licencjonowaniem UNIX-a – ostatnio również w Europie.

• • •

Najnowsza wersja UNIX-a z Berkeley nazywa się **4.3BSD Reno** i zawiera niektóre elementy spodziewanej mniej więcej za rok edycji 4.4. BSD. Najważniejsze nowości to mechanizmy sieciowe zgodne z ISO/OSI, implementacja Sun-owego NFS *(Network File System)*, oraz zgodność z POSIX-em.

• • •

Rywalizacja między **OSF/Motif** i **Open Look**, dwoma konkurencyjnymi propozycjami standardowego sprzętu graficznego (por. INFORMATYKA nr 9, 1990) chwilami przybiera humorystyczne formy. Najpierw w DEC ogłoszono zamiar przeniesienia OSF/Motif na komputery konkurencji. Wkrótce potem, na wystawie towarzystwa USENIX towarzyszącej konferencji, tępiony przez DEC Open Look reklamowano w stoisku firmy Sun za pomocą DECstation 3100.

• • •

W Open Software Foundation poinformowano ze skruchą, że zapowiadany od dawna OSF/Motif 1.1. nie zapewnia binarnej zgodności z OSF/Motif 1.0. Powodem jest przejście od X11R3 (poprzednia wersja X Window System) do X11R4. Konkurencji nie grożą te kłopoty: Open Look od początku korzysta z X11R4.

(JS)

MEDCOM

CHCESZ ZABEZPIECZYĆ BAZY DANYCH PRZED UTRATĄ INFORMACJI?

CHCESZ ZAPEWNIĆ PRAWIDŁOWĄ PRACĘ

KOMPUTERA * CENTRALI TELEFONICZNEJ * TELEFAKSÓW

W WARUNKACH ZANIKU NAPIĘCIA ZASILAJĄCEGO

kup **ZASILACZE BEZPRZERWOWE:**

naszej produkcji

UPS – 120 (abonenckie centrale telefoniczne i telefaksy)
UPS – 450
UPS – 850
UPS – 1450

produkcji tajwańskiej:
UPS – 500

Sprzedajemy również:

* AKUMULATORY BEZOBSŁUGOWE, HERMETYCZNE FIRMY PANASONIC do zasilania urządzeń alarmowych, jedyne dopuszczone do używania w pomieszczeniach zamkniętych,
* GENERATORY PRĄDOTWÓRCZE FIRMY ENDRESS O MOCY OD 1 kVA DO 350 kVA.

Twoje problemy w dziedzinie zasilania awaryjnego rozwiążą specjaliści z firmy MEDCOM.

BIURO
Al. Ujazdowskie 26/39
00-478 Warszawa
tel. 28-93-57, teleks 817060

ZAKŁAD PRODUKCYJNO-USŁUGOWY
ul. Bokserska 53
02-690 Warszawa

0/20/90

LET C = NOT C OR C.

Język pozwala na skorzystanie z wielu operatorów (jest ich 42), upraszczając zapis wielu standardowych konstrukcji algorytmów. Oczywiście bardzo polubiłem możliwość zapisu:

```
i++;  --i;  i+=5;  i*=2;  (* itp *)
```

W powiązaniu ze wskazywanym dostępem do struktur danych, uzyskujemy zapisy:

```
char*S, *t;                                          [KR, 118]
{                              /* kopiowanie napisu do znaku '\0'*/
  while (*s++ = *t++);
}
while ((s2[i]=s1[i]) I = '\0') ++i;                  [KR, 41]
if( q>=q→s.ptr && (p>q II p<q→s. ptr))               [KR, 196]
```

Tak, tylko przy tak dużej liczbie operatorów wyrażanych za pomocą symboli, uzyskano wiele niejednoznaczności, w interpretacji wyrażeń. Konieczne jest więc często uzupełnianie wyrażeń nawiasami. Jednocześnie postać niektórych operatorów może prowadzić do powstawania, groźnych, trudnych do wykrycia błędów, takich jak:

```
if ( !inword ) ...
if ( inword == 0 ) ...
if ( inword = 0) ...                                 [KR, 53]
```

wpisania operatora podstawienia '=' zamiast równości '==' i podobnie bitowych operatorów logicznych '&', 'I' zamiast zwykłych operatorów logicznych '&&', 'II'. Wykrycie takich pomyłek może być czasem dość czasochłonne, chociaż istnieją kompilatory wskazujące na podejrzane wyrażenia.

Ważną zaletą języka jest podobno możliwość operacji na wartościach bitowych, korzystając z operatorów przesuwania i operacji bitowych na zmiennych. Zachęcony tym, zastosowałem te operacje do prostej analizy określania liczby kolejnych jedynek w mapie bitowej.

```
byte x ;

for ( i=0; i<8 ; i++ )
{
x = (x<<1) ;
if ( x & 0200 ) then else ;
}
```

Algorytm tego rozwiązania był klasyczny i działał, tylko że dla dużych map zbyt długo. Po analizie, napisałem podprogram w asemblerze, który ten sam problem rozwiązywał za pomocą jednej instrukcji przesuwania i skoku warunkowego uzależnionego od wartości bitu C rejestru stanu. Zmiana ta spowodowała prawie czterokrotne przyspieszenie wykonywania programu. To mnie najbardziej zraziło do języka C (oczywiście w wydaniu Microsoft, wersja 5.0).

Najczęściej podnoszoną zaletą języka C jest zwartość tekstu programu. Zgoda, język nie jest przegadany, ale czy aby na pewno jest to zaleta? Normalnie pisany tekst programu w języku C jest nieczytelny. Duża liczba operatorów, okrutne separatory komentarzy oraz możliwość zapisania tych samych elementów składni na różne (równoważne sobie semantycznie) sposoby – powodują powstanie tekstu, który nawet dla jego autora po pewnym czasie staje się mało zrozumiały. Ograniczenie do jednopoziomowego definiowania podprogramów nie ułatwia rozróżnienia w modułach procedur i funkcji lokalnych od zewnętrznych.

Wiele pracy redakcyjnej i dodatkowych komentarzy wymaga uzyskanie tekstu programu mogącego być równocześnie jego dokumentacją. Moim zdaniem jest to ogromna wada języka eliminująca go w wielu zastosowaniach. Co ma bowiem zrobić inwestor dający duże pieniądze na opracowywanie programu, gdy po pewnym czasie zniknie mu któryś z programistów?

dokończenie z III s. okładki

No dobrze, powie ktoś, ale za to możemy otrzymać prawie tak efektywny kod wynikowy, jak napisany w asemblerze. A przecież czytelność programów asemblerowych jest dużo gorsza. Tak, zgoda, ale nie jestem taki przekonany, aby typowe programy zapisane w innych językach miały w działaniu dużo gorsze wyniki. Co więcej, często jest naprawdę mało istotne, czy program będzie realizował daną funkcję trzy czy cztery sekundy oraz zajmował nieco więcej i tak nie używanej pamięci operacyjnej. Znacznie ważniejszą sprawą jest możliwość poprawienia lub modyfikacji programu nawet po kilkunastu miesiącach eksploatacji, i to nie tylko przez jego autora. W tym przypadku ma sens podstawowe przeznaczenie języka C do realizacji oprogramowania systemowego. Nowe systemy operacyjne, translatory oraz inne tego typu programy mogą konstruować tylko silne firmy z wypracowanymi już metodami funkcjonowania. Pozostała masówka powstaje w dużym stopniu po partyzancku i wtedy dopiero ujawniają się kłopoty związane z pielęgnacją świeżego programu.

Pozostało nam jeszcze zagadnienie łatwej przenośności programów zapisanych w języku C. Faktem jest, że to dzięki C tak sprawnie rozprzestrzenia się system UNIX. Jak piszą twórcy C [KR, 14] „z 13 000 wierszy programu tylko około 800 – na najniższym poziomie – napisano w asemblerze". Tak ale był to system operacyjny oraz związane z nim oprogramowanie wspomagające. Niestety, pomimo zaawansowanych prac nad standardem języka, poszczególne jego implementacje nieco się różnią, szczególnie w zakresie standardowych podprogramów bibliotecznych. To „nieco" jest wystarczająco duże, aby przeniesienie programu z jednego typu komputera na inny stało się złożonym procesem. A jeszcze omawiana wcześniej słaba czytelność tekstu programu nie wspomaga tego przedsięwzięcia. Sam byłem świadkiem, jak mozolne były prace nad przeniesieniem stosunkowo prostego programu napisanego w języku Microsofr C dla IBM PC na Macintosha z translatorem Think C.

Punktując język C nie przekreślam jego ważnej roli w rozwoju technologii programowania. Nie należy jednak sądzić, że będzie to ostatni język, jakiego będą się uczyć rzesze programistów. Nawet jego kolejne mutacje, jak C++, ParC itp. nie zmieniają mojej opinii, że należy spodziewać się w najbliższych latach popularności kolejnego języka programowania. Czy będzie to któryś z już znanych istniejących języków, czy już istniejący w wąskim gronie akademickim lub przemysłowym – trudno powiedzieć. Postaram się powrócić do tego problemu – oczekując również innych opinii.

WACŁAW ISZKOWSKI

P.S.1. Większość przykładów podanych w tekście zaczerpnąłem z tłumaczenia klasycznej książki B. W. Kernighama i D. M. Ritchie, „Język C", wydanej przez WNT, 1988 (oznaczenie [KR, nr strony] dotyczy tej właśnie pracy).

P.S.2. Zbitki poglądów i spostrzeżeń, tutaj prezentowanych, nie są oficjalnym stanowiskiem Redakcji ani żadnej innej organizacji. Ich prezentacja ma służyć pobudzeniu dyskusji na dany temat nawet przez wzburzenie moich ewentualnych adwersarzy.

Wszystkich zainteresowanych nabyciem INFORMATYKI prosimy o kontakt z redakcją (tel. 39-14-34) lub Działem Handlowym SIGMY (tel. 40-30-86) – możemy wysyłać egzemplarze za zaliczeniem pocztowym. Czytelnicy z terenu Warszawy mogą kupić poszczególne numery w Klubie SIGMY przy ul. Mazowieckiej 12, w księgarni PP „Domu Książki" przy ul. Mokotowskiej 51/53 oraz w lokalu redakcji.

Jednocześnie przypominamy, że najpewniejszą formą otrzymywania INFORMATYKI jest prenumerata.

Kulikowski J.L.: Komputerowa analiza obrazów w biologii i medycynie

INFORMATYKA 1991, nr 1, s. 1

Charakterystyka światowego rozwoju oraz stan obecny rozwiązań w dziedzinie komputerowego przetwarzania obrazów dla potrzeb diagnostyki lekarskiej i badań biologicznych, a także prac prowadzonych w Polsce.

Kulikowski J.L.: Computer analysis of images in biology and medicine

INFORMATYKA 1991, No. 1, p. 1

Characteristics of world development and actual situation of solutions in computer image processing for medical diagnostics and biological research, as well as of polish scientific research in this field.

Kulikowski J.L.: Computeranalyse von Bildern in Biologie und Medizin

INFORMATYKA 1991, Nr. 1, S. 1

Eine Charakteristik von Entwicklung und jetzigem Stand der Lösungen im Bereich der Computerbildverarbeitung zur medizinischen Diagnostik und Biologieforschung, sowie von in Polen geführten wissenschaftlichen Arbeiten.

Skaba W.: Współczesne metody kompresji danych: teoria, algorytmy i zastosowania

INFORMATYKA 1991, nr 1, s. 6

Podstawy teoretyczne oraz charakterystyka współczesnych metod, algorytmów i zastosowań kompresji danych.

Skaba W.: Contemporary data compression methods: theory, algorithms and applications

INFORMATYKA 1991, No. 1, p. 6

Theoretic foundations and characteristics of contemporary methods, algorithms and applications of data compression.

Skaba W.: Zeitgenössische Datenverdichtungsmethoden: Theorie, Algorithmen und Anwendungen

INFORMATYKA 1991, Nr. 1, S. 6

Theoretische Grundlagen und eine Charakteristik der zeitgenössischen Methoden, Algorithmen und Anwendungen der Datenverdichtung.

Fuglewicz P.: Systemy CASE – problemy, techniki, rozwiązania (2)

INFORMATYKA 1991, nr 1, s. 14

Druga część przeglądu metod i rozwiązań CASE wspomaganej komputerowo inżynierii oprogramowania, zawierająca charakterystykę sześciu opracowanych w Polsce programów z tej dziedziny.

Fuglewicz P.: CASE systems – problems, techniques and solutions (2)

INFORMATYKA 1991, No. 1, p. 14

Second part of the CASE Computer Aided Software Engineering methods and solutions, which includes characteristics of the six polish programs in this field.

Fuglewicz P.: CASE-Systeme – Probleme, Techniken und Lösungen (2)

INFORMATYKA 1991, Nr. 1, S. 14

Zweiter Teil der Übersicht von CASE-Methoden und Lösungen, der eine Charakteristik von den sechs in Polen erarbeiteten Programmen in diesem Bereich umfasst.

Ceny ogłoszeń w 1991 r.

Wzrost kosztów wydania INFORMATYKI powoduje, że w 1991 r. musi również nastąpić niezbędne skorygowanie cen ogłoszeń. Pragniemy podkreślić, że mimo 20% podwyżki, ceny te będą nadal znacznie niższe niż w innych czasopismach, a mianowicie:

- cała strona 600 000 zł
- 3/4 strony 540 000 zł
- 2/3 strony 480 000 zł
- 1/2 strony 360 000 zł
- 1/3 strony 240 000 zł
- 1/4 strony 180 000 zł
- 1/8 strony 120 000 zł
- poniżej 1/8 strony 2 500 zł za centymetr kwadratowy

Dopłaty
- za każdy dodatkowy kolor +30%
- za I i IV stronę okładki +100%
- za II i III stronę okładki +50%

Rabaty
- za powtórzenie 3–6 razy – 10%
- za powtórzenie ponad 6 razy – 20%
- za teksty o charakterze informacji techniczno-handlowej (artykuły) – 50%

Powyższe ceny uwzględniają wszystkie koszty przygotowania ogłoszenia lub informacji techniczno-handlowej do druku.

W przypadku rezygnacji z ogłoszenia przed przekazaniem do druku, Zamawiający ponosi koszty w wysokości 25% wartości złożonego, zamówienia. Przy rezygnacji po przekazaniu do druku, zamawiający ponosi pełny jego koszt. Zmiana ustalonej przy zamówieniu treści ogłoszenia po przekazaniu do druku powoduje zwiększenie należności o 20%, lecz tylko za jedno ogłoszenie (koszt wykonania nowej kliszy). Ze względu na znaczny napływ zamówień w drugim półroczu br., zachęcamy do wcześniejszej rezerwacji miejsc ogłoszeń. Po zgłoszeniu telefonicznym skierujemy do Państwa naszego przedstawiciela, który udzieli wyczerpujących informacji, doradzi i dokona szczegółowych uzgodnień technicznych.

Zapewniamy szybką i terminową realizację ogłoszeń!

Nasz adres: INFORMATYKA, 01–552 Warszawa, pl. Inwalidów 10, p. 104, 105, tel. 39-14-34.

Wskazówki dla Autorów

Nadsyłane artykuły nie mogą być publikowane lub przeznaczone do opublikowania w innych czasopismach.

Materiał oprócz tekstu zasadniczego powinien zawierać:
* krótki życiorys zawodowy Autora i jego zdjęcie,
* wykaz literatury,
* tabele,
* materiał ilustracyjny (rysunki, zdjęcia czarno-białe, wydruki) dołączony do artykułu (nie wklejać materiału w tekst),
* podpisy pod ilustracje.

Na osobnej stronie prosimy podać tytuł naukowy, imię i nazwisko, nazwę zakładu pracy, adres domowy (koniecznie!), numer telefonu oraz informację, jaką drogą przekazać honorarium – kasa Wydawnictwa, poczta, bank.

Tekst artykułu prosimy dostarczyć w formie maszynopisu lub wydruku komputerowego, pisany jednostronnie, z podwójnym odstępem – bardzo ważne! – czyli 30 wierszy na stronie i 60 znaków w wierszu.

Wykaz literatury w porządku alfabetycznym.

Tabele – każda na osobnej stronie – powinny być ponumerowane i opatrzone tytułem oraz ściśle związane z tekstem (zaznaczone miejsce tabeli w tekście).

Rysunki – winny być czytelne (zaznaczyć ich miejsce w tekście), mogą być wykonane ołówkiem.

Zdjęcia – czarno-białe, kontrastowe, na błyszczącym papierze.

Wydruki – czytelne, kontrastowe, wykonane na białym papierze. Format – maksymalnie 18 cm w podstawie.

Każde pierwsze słowo opisu rysunków powinno być pisane dużą literą.

Podpisy pod ilustracje powinny być napisane na oddzielnej stronie, oprócz kolejnego numeru powinny zawierać tytuł (rysunku, zdjęcia, wydruku) i ew. legendę dotyczącą poszczególnych elementów.

Autor opublikowanego w INFORMATYCE artykułu otrzymuje honararium i bezpłatny egzemplarz okazowy.

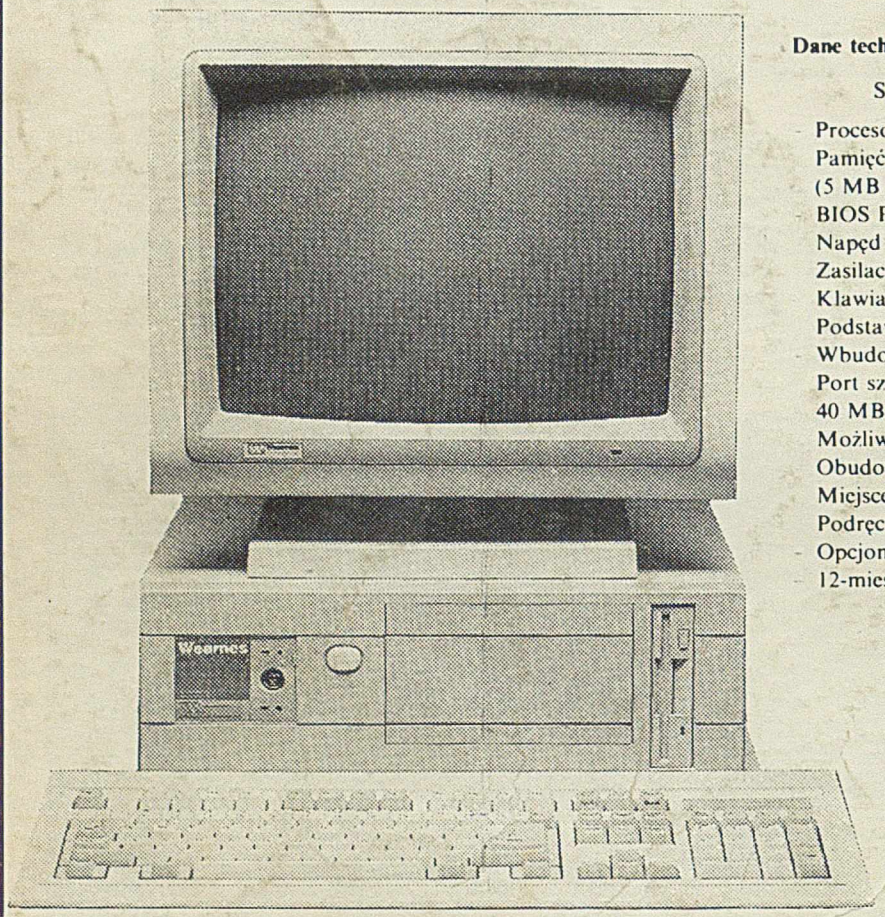
Materiałów nie zamówionych redakcja nie zwraca.

KOMPUTER
Z WBUDOWANĄ PRZYSZŁOŚCIĄ

Komputer zaprojektowany przez czołowego światowego producenta mikrokomputerów – firmę ADVANCED LOGIC RESEARCH i wyprodukowany przez firmę WEARNES TECHNOLOGY.

Komputer, którego możliwości i cena oszałamiają konkurencję.

Komputer, który nie zestarzeje się, dzięki możliwości rozbudowy do architektury 386SX i 486.



Dane techniczne:

SERIA WEARNES BOLDLINE „M"

- Procesor 80286 12.5 Mhz
- Pamięć 1 MB RAM, możliwość rozbudowy do 16 MB (5 MB na płycie głównej)
- BIOS Phoenix
- Napęd dyskietek 3,5" 1,44 MB
- Zasilacz 110 Watt
- Klawiatura 101 klawiszy
- Podstawka dla koprocesora matematycznego 80287
- Wbudowany sterownik dyskowy z przeplotem 1:1
- Port szeregowy i równoległy
- 40 MB dysk sztywny
- Możliwość korzystania z EMS 4.0
- Obudowa typu „compact"
- Miejsce na dwa napędy 5,25" o wysokości 1/2
- Podręcznik i dyskietka z programem konfiguracyjnym
- Opcjonalna rozbudowa do 386SX i 486
- 12-miesięczna gwarancja

Komputer z wbudowaną przyszłością!

BOLDLINE COMPUTERS
GRUPA MICOMP-TECH
Biuro Informacji Techniczno-Handlowej
ul. Astrów 7, 40–045 Katowice
telefon i telefaks: 518-628
teleks: 315687 COMP PL

DYSTRYBUTORZY:
PTH „TECHMEX"
43–300 Bielsko-Biała
ul. Curie-Skłodowskiej 13
tel.: 42-198, 47-555, telefaks: 47-624,
teleks: 35325

EO/1123/89